IBM RT Advanced Interactive Executive Operating System Version 2.2

Managing the AIX Operating System

Programming Family

IBM

SC23-2008-1

IBM RT Advanced Interactive Executive Operating System Version 2.2

Managing the AIX Operating System

Programming Family

IBM

SC23-2008-1

Second Edition (September 1988)

Portions of the code and documentation described in this book were developed at the Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California under the auspices of the Regents of the University of California.

The Rand MH Message Handling System was developed by the Rand Corporation and the University of California.

This edition applies to Version 2.2 of the IBM AIX Operating System. The previous edition still applies to Version 2.2 of the AIX Operating System and can still be ordered using order number SBOF-0168. Changes are made periodically to the information herein; these changes will be reported in technical newsletters or in new editions of this publication.

About This Book

This book explains how to manage the AIX system. ***System management*** is a broad term for all of the tasks required to adapt the AIX Operating System to your needs and to keep the system in good working order. Among the topics included are:

- Setting up user accounts of different types
- Creating, using, and maintaining file systems
- Making backup copies of information stored on the system
- Using system accounting facilities.

Who Should Read This Book

This book is written for those who are responsible for managing an AIX system. If more than one person uses your system, system management responsibilities may be given to one person or shared among several. If you are the only person using your system, you still must perform certain system management tasks.

Before You Begin

Before you can begin to work with this book, your RT system must be set up and the AIX Operating System must be installed. In addition, you should have a **user name** and possibly a password. If your system is not installed, see *Installing and Customizing the AIX Operating System.* If you need a user name, see "Creating, Changing, and Removing Accounts—The users Command" on page 2-13.

Note: Many of the tasks in this book require you to use one of the RT text editing programs. The following editing programs are available on the RT system:

- **ed** (see *Using the AIX Operating System*)
- **INed**[1] (see *INed*)
- **vi** (see *AIX Operating System Commands Reference*)

How to Use This Book

This book is divided into chapters, with each chapter devoted to a major AIX Operating System concept or feature:

- Chapter 1, "Introduction to System Management" on page 1-1 includes background information necessary for effective system management. It explains the structure of the system in general, and the file system in particular, from the viewpoint of someone responsible for managing and maintaining the AIX Operating System.
- Certain system management tasks must be performed routinely. Chapter 2, "Routine System Management" on page 2-1 explains such common system management tasks as starting the system, establishing user accounts, creating and using file systems, and making backup copies of data stored in file systems.
- In addition to the routine system management tasks, there are other tasks that you may need to perform periodically, as well as other concepts that you should understand in order to make your system as efficient and dependable as possible.
Chapter 3, "Maintaining the AIX Operating System" on page 3-1 explains several of these system maintenance tasks, including the very important task of maintaining the file system. This chapter also contains explanatory information about the AIX Operating System input/output, queueing, and error-reporting systems.
- Chapter 4, "Additional System Management Topics" on page 4-1 is an assortment of other useful system management information. Among the topics in this chapter are how to run commands at pre-determined times, managing display station features, managing printers, and maintaining system performance. You may find it helpful to simply browse through this chapter, looking for features or capabilities that you can use to make your system better suit your needs.
- Chapter 5, "Managing Multiuser Systems" on page 5-1 describes the AIX Operating System accounting system, which allows you to collect data about how your system is being used, and explains the AIX facilities for measuring and reporting input and output, processing unit utilization, file system access, and other system activity. This chapter also describes some of the AIX facilities for communicating with system users.

---

1 INed is a registered trademark of INTERACTIVE Systems Corporation

- The system manager must maintain the integrity of the system. The controlled access mode is designed to provide several useful tools for managing system security. Chapter 6, "Managing System Security" on page 6-1 provides information about managing various aspects of security that comprise the controlled access mode. This chapter includes discussions about access control, the identification and authentication procedure, the trusted computing base, and system auditing.
- The rest of this book is devoted to the management and maintenance of AIX data communications facilities. The chapters are:
  - Chapter 7, "Managing the Electronic Mail System" on page 7-1
  - Chapter 8, "Managing Asynchronous Terminal Emulation (ATE)" on page 8-1
  - Chapter 9, "Managing the Basic Networking Utilities" on page 9-1
  - Chapter 10, "Overview of the Message Handling Package" on page 10-1
  - Chapter 11, "Managing Distributed Services" on page 11-1
  - Chapter 12, "Managing the IBM AIX/RT Network File System" on page 12-1.

In addition, this book also includes supplemental information in the appendixes, and a glossary and an index to make it easier for you to find information.

A Reader's Comment Form and Book Evaluation Form are provided at the back of this book. Use the Reader's Comment Form at any time to give IBM information that may improve the book. After you become familiar with the book, use the Book Evaluation Form to give IBM specific feedback about the book.

You can use this book in one of two ways:

- As a training manual. Read and work through it from the first chapter to the last one. This should give you a general understanding of the AIX Operating System.
- As a reference manual. Use the "Contents" and "Index" to locate particular topics. This is a good way to refresh your memory or learn more details about the AIX Operating System.

Special Features

This book uses type style to distinguish among kinds of information. General information is printed in the standard type style (the type style used for this sentence). The following type styles indicate other types of information:

New terms

Each time a new term is introduced, its first occurrence is printed in this type style (for example, "the AIX Operating System ***file system***").

System parts

The names for keys, commands, files, and other parts of the system are printed in this type style (for example, "the **cp** command").

Variable information
The names for information that you must provide are printed in this type style (for example, "type *yourname*").

`Special characters`
Any characters that have a special meaning are printed in this type style (for example, "the `&` and `&&` operators have different uses"). This type is also used for the names of files that you create as you work through this book (for example, "create a file named `afile`").

`Information you are to type`
Many examples in this book are designed for you to try them on your own system; the information that you should type is printed in this type style (for example, "type `ls text` and press **Enter**").

Where appropriate, the chapters and major sections of this book begin with a box containing quick reference material, for example:

To Use a Quick Reference Box

1. Skip the quick reference boxes the first time you read a section.
2. Use the quick reference boxes as a fast path through the book.
3. Refer to the quick reference boxes to refresh your memory.

You should use the boxes for reference after you are generally familiar with the contents of a section or chapter. You can skip the box the first time you read a section. The boxes make a convenient ***fast path*** through the book, but they are not comprehensive and they are not intended to take the place of the explanatory material in each section.

After the quick reference boxes, each chapter in this book takes the same general approach to the topics it covers—a series of explanations and examples. The examples build upon each other; in many instances, an example uses a file created in a previous example. Therefore, if you intend to follow the examples on your system, it is important for you to work through each chapter from start to finish.

In the examples, the characters you should type are printed in blue, as this example shows:

```
$ ls
  afile
  bfile
  cfile
$ _
```

After you type the characters on a line, press the **Enter** key.

In the text, whenever you are told to ***enter*** a command or other information, you should type the information and then press the **Enter** key.

Related Books

- *IBM RT Installing and Customizing the AIX Operating System* provides step-by-step instructions for installing and customizing the AIX Operating System, including how to add or delete devices from the system and how to define device characteristics. This book also explains how to create, delete, or change AIX and non-AIX minidisks.
- *IBM RT Using the AIX Operating System* describes using the AIX Operating System commands, working with file systems, developing shell procedures, and using data communications facilities.
- *IBM RT AIX Operating System Commands Reference* lists and describes the AIX Operating System commands.
- *IBM RT Guide to Operations* describes the IBM 6151 and IBM 6150 system units, the displays, keyboard, and other devices that can be attached. This guide also includes procedures for operating the hardware and moving the IBM 6151 and IBM 6150 system units.
- *IBM RT Problem Determination Guide* provides instructions for running diagnostic routines to locate and identify hardware problems. A problem determination guide for software and three high-capacity (1.2MB) diskettes containing the IBM RT diagnostic routines are included.
- *IBM RT Usability Services Guide* shows how to create and print text files, work with directories, start application programs, and do other basic tasks with Usability Services. (Packaged with *Usability Services Reference*)
- *IBM RT Usability Services Reference* supplements *IBM RT Usability Services Guide* by including information on using all of the Usability Services commands. (Packaged with *Usability Services Guide*)
- *IBM RT Exploring Usability Services* is an online tutorial for first-time users of the Usability Services. This tutorial simulates the user interface and shows how to use the keyboard and the optional mouse, how to manipulate windows, and how to use files and directories.
- *IBM RT Messages Reference* lists messages displayed by the IBM RT and explains how to respond to the messages.
- *IBM RT Using DOS Services* provides step-by-step information for using AIX Operating System shell. (Available optionally; packaged with *IBM RT DOS Services Reference*)
- *IBM RT AIX Operating System Technical Reference* describes the system calls and subroutines that a C programmer uses to write programs for the AIX Operating System.

This book also includes information about the AIX file system, file formats, special files, GSL subroutines, and the sockets interprocess communications facility. (Available optionally)

- *IBM RT INed* provides guide and reference information for using the INed program to create and revise files.
- *IBM RT AIX Operating System Programming Tools and Interfaces* describes the programming environment of the AIX Operating System and includes information about using the operating system tools to develop, compile, and debug programs. In addition, this book describes the operating system services and how to take advantage of them in a program. This book also includes a diskette that includes programming examples, written in C language, to illustrate using system calls and subroutines in short, working programs. (Available optionally)
- *IBM RT SNA Services Guide and Reference* describes the capabilities and functions provided by the IBM RT Systems Network Architecture (SNA) Services when it is installed on an IBM RT. It includes information on the structure and operation of SNA (Systems Network Architecture) on the RT, configuring a network, controlling SNA on the local system, finding problems associated with SNA Services, programming using the SNA Services application programming interface, and using the Usability Services extensions that are active when SNA Services is installed. (Available optionally)
- *IBM RT Interface Program for use with TCP/IP* describes the Interface Program commands for transferring data among host computers, logging into remote computers, and managing networks. This book also describes the programming interfaces to the Interface Program.

Ordering Additional Copies of This Book

To order additional copies of this publication (without program diskettes), use either of the following sources:

- To order from your IBM representative, use Order Number SB0F-1810.
- To order from your IBM authorized RT dealer, use Part Number 27F4353.

A binder, and the *Managing the AIX Operating System* manual are included with the order. For information on ordering the binder and manual separately, contact your IBM representative or your IBM authorized RT dealer.

Contents

Part 1. Managing the AIX Operating System

Chapter 1. Introduction to System Management

CONTENTS

About this Chapter

The AIX Operating System is a powerful and versatile tool. Generally, you use the operating system to do your work—for example, to process data, edit text files, run spreadsheet programs, and communicate with other users. There are times, however, when you must work on the operating system itself—for example, to add new users to the system or to create, back up, and repair file systems. The work you do on the operating system is called ***system management***.

This chapter includes the background information necessary for effective system management. While this chapter and the remainder of this book provide much specific information and careful guidance, you should understand that the job of system management is not the same on any two systems, and that system management requirements may change considerably on a particular system over time. Thus, in addition to becoming familiar with the information in the rest of this book, the best way to manage your system effectively is to continue to learn about the AIX system and the requirements placed on your system by its users.

General System Structure

As shown in Figure 1-1, the AIX Operating System is comprised of the kernel, the Virtual Resource Manager, shells, libraries, and various subsystems. The AIX components are interdependent; the combined function and features of the components form the AIX Operating System.

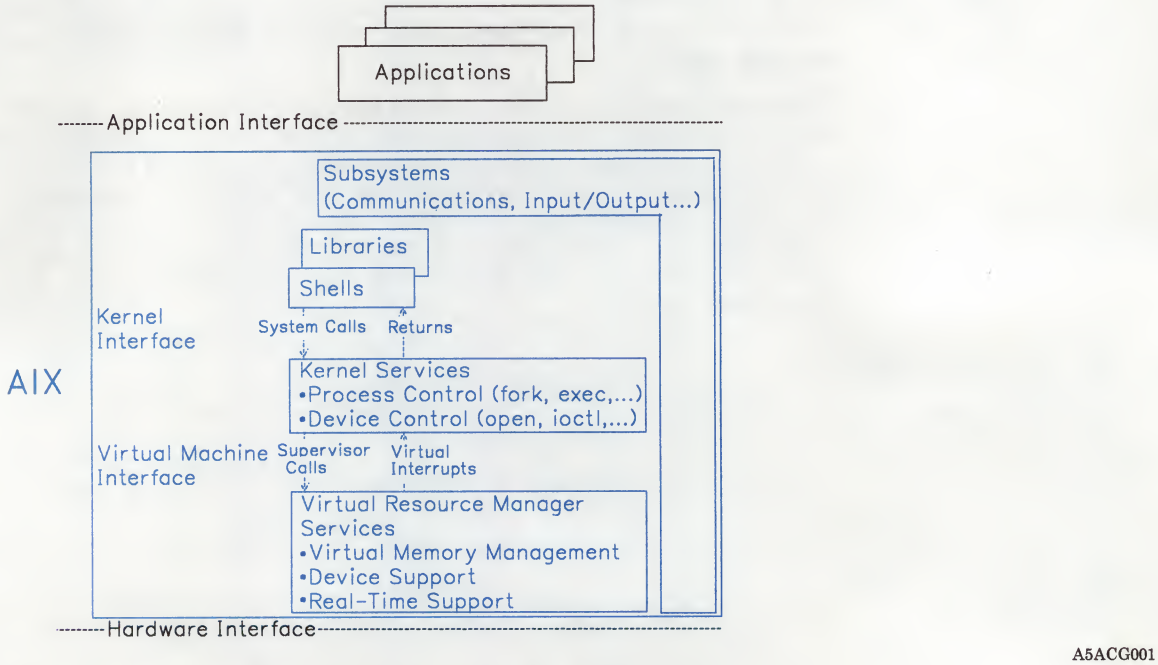


Figure 1-1. Parts of the AIX Operating System

The Virtual Resource Manager (VRM)

The VRM is a portion of the AIX Operating System that provides various services, interfaces, and run-time routines through which AIX controls the IBM RT hardware and peripherals.

The Kernel

A part of the AIX Operating System participating in the control of the computer functions, such as input/output, management and control of hardware/software, and the scheduling of user processes.

The Shell

The shell, often called an ***interface*** or a ***command interpreter***, is the part of the operating system that makes it possible for you to use the kernel. The shell is actually an ordinary program that is said to run *on top of* the kernel.

You can use different interfaces, or different versions of the standard shell. Following is a list of the interfaces available on the AIX system:

- AIX shell
- Usability Services, described in *Usability Services Guide*
- DOS Services, described in *Using DOS Services*
- C Shell, described under **csh** in *AIX Operating System Commands Reference.*

The information in this book is about the standard AIX shell.

The shell makes it possible for other programs to run by starting kernel processes. The kernel manages how its services are shared among programs.

The shell also makes it convenient for you to use certain basic kernel services, such as:

- Device-independent **input** and **output** (used by the shell to accomplish I/O redirection; for more information about redirecting the input and output, see *Using the AIX Operating System.*)
- **Pipes** (For information about pipes, see *Using the AIX Operating System.*)

Finally, the shell is a ***command programming language***. That is, with the shell, you can develop your own commands or shell procedures without having to use a conventional programming language. (For more information about the shell, see *Using the AIX Operating System* and **sh** in *AIX Operating System Commands Reference.*)

The File System—Background for System Management

Note: Before you begin this section, you should be familiar with how the file system appears to an ordinary system user. For information on file systems, see *Using the AIX Operating System.*

A file system is a complete directory structure, including a root directory and any subdirectories and files beneath it. File systems are confined either to a single ***minidisk*** (partition of a fixed disk) or to a diskette (one file system per diskette). Some of the most important system management jobs have to do with file systems, specifically:

- Allocating space for file systems on minidisks
- Creating file systems
- Making file system space available to system users
- Monitoring file system space usage
- Backing up file systems to guard against data loss in the event of system or disk failures
- Maintaining file systems in a consistent state.

There are certain system commands designed specifically for system management. Of those commands, the ones you probably will use regularly for working with file systems are:

backup Performs a full or incremental backup of a file system.

clri Clears i-nodes (used when file system inconsistencies cannot be corrected by **fsck**).

dd Copies data directly from one device to another (for making file system backups).

df Reports the amount of space used and free on a file system.

fsck Checks file systems and repairs inconsistencies.

mkfs Makes a file system of a specified size on a specified minidisk.

mount Attaches a file system to the system-wide naming structure so that files and directories in that file system can be accessed.

restore Restores files from a backup.

umount Removes a file system from the system-wide naming structure, making the files and directories in the file system inaccessible.

Regardless of the device they reside on (diskette or minidisk), all file systems have the same structure. Thus, you can move a file system from one device to another, provided the destination device is large enough. Disk space is allocated for file systems in ***blocks*** that can contain 512 ***bytes*** of data.

A file system has four major parts:

- Bootstrap block
- Superblock
- I-node blocks
- Data blocks.

Figure 1-2 represents the major parts of a file system.

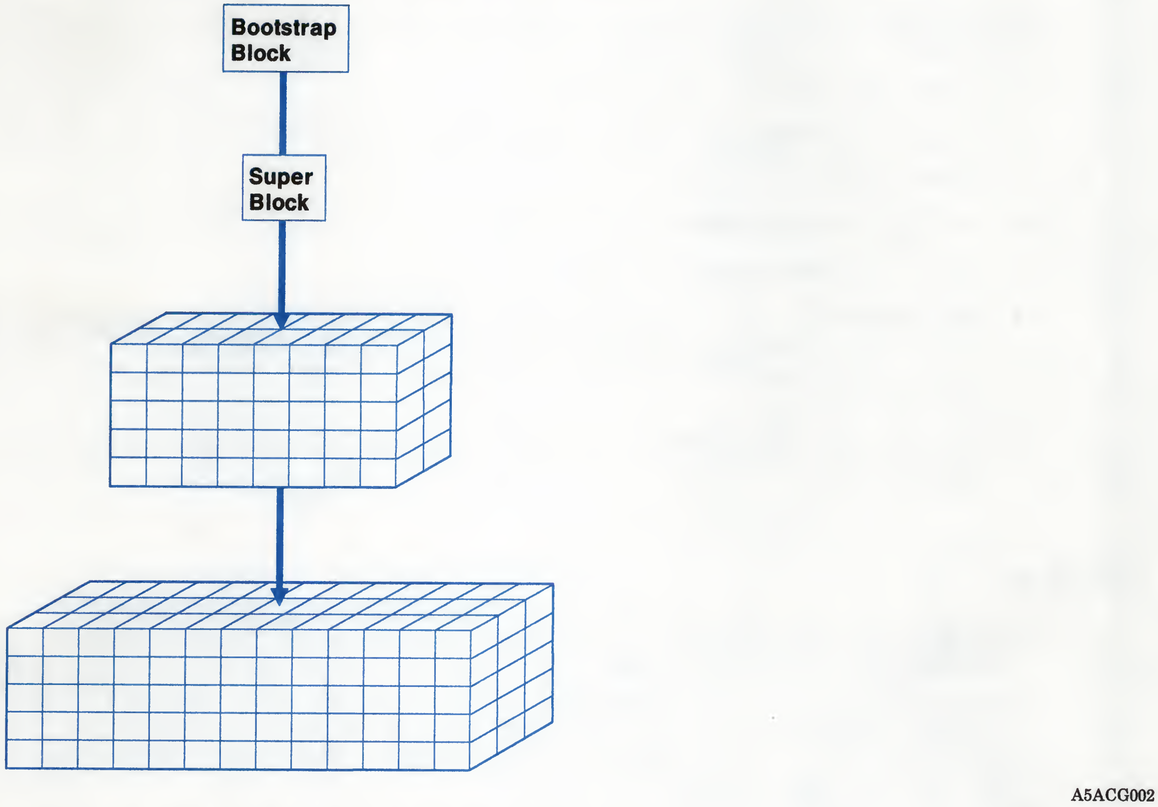


Figure 1-2. Major Parts of an AIX File System

Bootstrap Block

The first block of every file system (block 0) is reserved for a ***bootstrap***, or initialization, program. The bootstrap block is not actually part of the file system structure. File system data begin on block 1 of the minidisk.

The Superblock

Block 1 of every file system is called the ***superblock***. Among the most important information the superblock contains are:

- Total size of the file system (in blocks).
- Number of blocks reserved for i-nodes (explained below).
- Name of the file system.
- Volume ID of the minidisk or diskette.
- Date of the last superblock update.
- The head of the ***free-block list***, a chain that contains all of the free blocks in the file system (the blocks available for allocation). When new blocks are allocated to a file, they are allocated from this list. When a file is deleted, its blocks are returned to this list.
- A list of free i-nodes. This is a partial list of i-nodes available to be allocated for newly created files.

I-nodes

After the superblock there is a group of blocks that contain ***i-nodes*** (descriptions of the individual files in the file system). There is one i-node for each possible file in the file system. Each i-node is associated with an i-node number, or i-number. For any file system, there is a maximum number of i-nodes and thus, a maximum number of files that the file system can contain. This maximum number varies with the size of the file system. The first i-node (i-node 1) on every file system is unnamed and unused. When associated with files, the remaining i-nodes contain the following information:

- **File type**. Possible file types are ordinary file, directory, block device, character device, and first-in-first-out (also sometimes called *FIFO* or *named pipe*).
- **File owner**. The i-node contains the user and group IDs associated with the file.

- **Protection information**. This is a specification of read, write, and execute access for the owner, members of the group associated with the file, and others. It also includes other mode information specified by the **chmod** command. (For a discussion of protections, see *Using the AIX Operating System.*)
- **Link count**. A directory entry (link) consists of a name and the number of the i-node that represents the file (its ***i-number***). The link count specifies the number of directory entries that refer to the file. A file is deleted when the link count becomes zero. That is, its i-node is returned to the list of free i-nodes and its associated data blocks are returned to the free-block list. (For a discussion of links, see *Using the AIX Operating System.*)
- **Size of the file (in bytes)**.
- **Date the file was last accessed**.
- **Date the file was last modified**.
- **Date the i-node was last modified**.
- **Pointers to data blocks**. These pointers indicate the location of the data blocks on the physical disk.

I-node 2 must correspond to the root directory for the file system. All other files in the file system are below the root directory in the file system. Beyond i-node 2, any i-node can be assigned to any file. Similarly, any data block can be assigned to any file. Neither i-nodes nor blocks have to be allocated in any particular order.

Data Blocks

Beyond the i-nodes, the file system consists of ***data blocks***. Each data block can contain 512 bytes of information. The i-node points directly to the first 10 data blocks of the file.

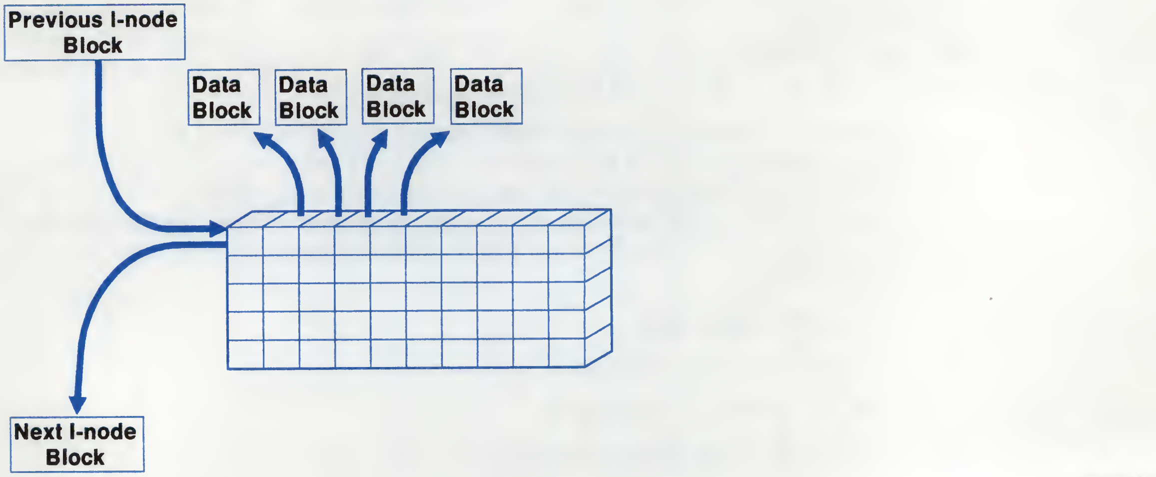
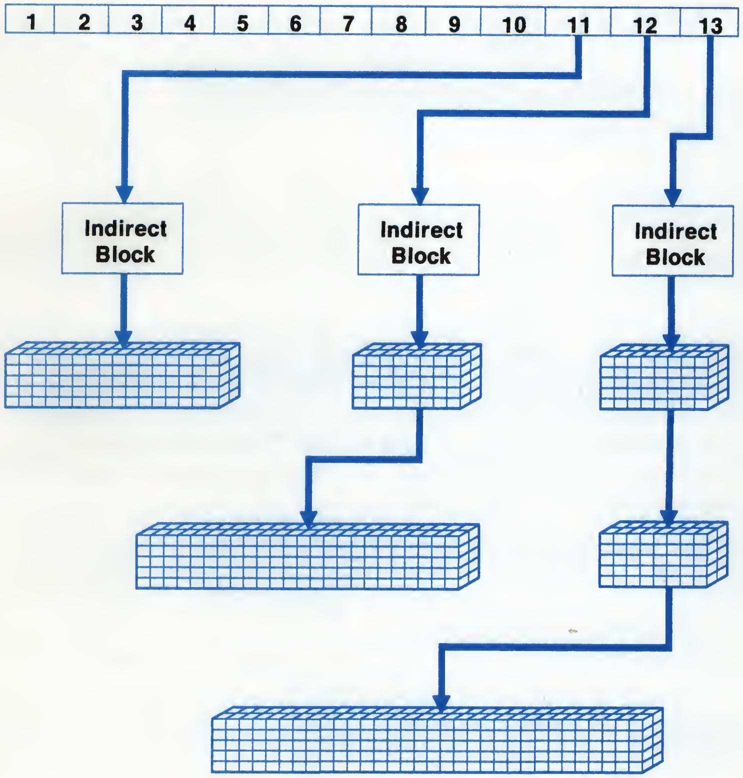


Figure 1-3. Direct Pointers from an I-node to Data Blocks

When a file contains more than 10 blocks of data, block number 11 is an ***indirect block***. The indirect block contains ***pointers*** to 128 additional data blocks. A file that has one indirect block is called ***single-indirect***, and has a maximum size of 139 blocks (the first 10 data blocks, plus the indirect block, plus the 128 data blocks).

If a file is larger than 138 blocks, block number 12 is a ***double-indirect*** block. A double-indirect block points to an indirect block that contains 128 pointers to other indirect blocks. Each of those indirect blocks in turn points to 128 data blocks. Thus, a double-indirect file has a maximum size of 138 + (128 * 128), or 16,522 blocks.

The largest files possible are ***triple-indirect***. Block number 13 of a triple-indirect file points to the first of three levels of indirect blocks, allowing a maximum file size of 16,522 + (128 * 128 *128), or 2,113,674 blocks. Figure 1-4 shows the relationship of data blocks and indirect blocks associated with an i-node.



A5ACG004

Figure 1-4. The Relationship of Data Blocks and Indirect Blocks

The Base AIX File System

The AIX operating system is made up of five separate file systems:

/ (called ***root***)

/usr

/tmp

/u

/vrm

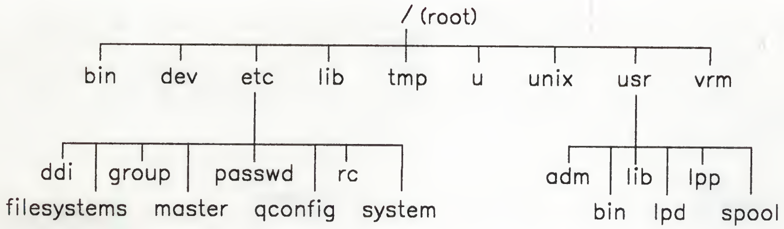
Each of these file systems resides on a separate minidisk. The system automatically mounts all five file systems when it initializes. Under normal conditions, it does not matter that there are five file systems—they mount automatically and function like a single file system. However, multiple file systems are convenient for certain system management tasks (for example, backing up, restoring, and repairing file systems) because they allow you to isolate a part of the system while you work on it.

In addition to the file system minidisks, there are two special purpose minidisks:

- Page space
- Dump.

Major Files and Directories

The AIX Operating System contains hundreds of files and directories, even before anyone begins to work on the system. (If you are curious about just how many files the operating system contains, enter the command `ls -Ra` to display the path name of every file in the system.) You do not need to work directly with most of these, or even to know where they are located in the file system. There are a few files and directories, however, that are important in routine system management. You should be familiar with their contents and location. Figure 1-5 on page 1-13 shows the position of these files and directories in the file tree.



A5ACG005

Figure 1-5. The Base AIX File System

Following are brief descriptions of the major AIX system files and directories:

/
: The highest directory in the file system hierarchy, usually called the ***root*** directory.

/bin
: A directory containing the basic commands required to run the system. Additional commands are contained in **/usr/bin** and **/usr/lib**.

/dev
: A directory containing the special files that allow input and output operations to system devices (such as fixed disks, diskette drives, and terminals). (For more information on special files and system devices, see "The Input/Output System" on page 3-13.)

/etc
: A directory containing most of the commands required for system startup and maintenance.

/etc/security
: A directory containing security configuration files.

/lib
: A directory containing files that make up the C language programming library and parts of the C compiler.

/tmp
: A directory containing temporary files created by processes.

/u
: A directory containing the home directories of all system users.

/unix
: A file containing the operating system kernel.

/usr
: A directory containing other directories of commands.

/vrm
A directory containing files and other directories associated with the VRM.

/etc/ddi
A directory containing files that describe system devices (for example, printers). **ddi** stands for **device dependent information**.

/etc/filesystems
A file composed of stanzas that describe file systems.

/etc/group
A file, which in conjunction with **/etc/security/group**, contains the information that defines groups of users for the system.

/etc/master
A file that contains information about system devices and that is used by the **config** program to create configuration files.

/etc/passwd
A file, which in conjunction with **/etc/security/passwd** contains the information that defines users for the system.

/etc/qconfig
A file that describes the available queues and devices and is referred to by the **print** command and the **qdaemon** process.

/etc/rc
A file containing the system startup routine (*run commands*). The usual way to change how the system starts up (for example, to specify what programs run automatically) is to modify this file.

/etc/security/config
A file containing stanzas that configure security subsystems.

etc/security/group
A file, which in conjunction with **/etc/group**, contains information about group passwords (for example, the encrypted password and the audit class or classes given to a group).

etc/security/passwd
A file, which in conjunction with **/etc/passwd**, contains information about user passwords (for example, the encrypted password and the audit class or classes assingned to a user).

/etc/security/sysck.cfg
A file containing stanzas that describe correct configurations of system files.

/etc/system
A file that contains descriptions of the devices that make up the current system.

/usr/adm

A directory containing the commands used in system accounting operations.

/usr/bin

A directory containing commands typically available to all system users and not contained in **/bin**.

/usr/lib

A directory containing system libraries and system management, text processing, and other commands not found in **/usr/bin** or **/bin**.

/usr/lpd

A directory containing commands used to send data to system printers and to collect accounting statistics on printer usage.

/usr/lpp

A directory that contains the directories and files, including history files, which are associated with installed licensed programs.

/usr/spool

A directory containing the ***spool***ed files to be sent between users, between the system and devices, and between systems.

Finding and Viewing System Files

As you learn about the AIX system, you probably will find it very helpful to be able to locate directories and files and look at their contents. The following three commands make it easy for you to do so:

find

Use the **find** command to search the file system for a file (or directory) when you know its name. The command:

find / -name *filename* -print

searches the file system, from the root directory (/) down, for *filename*. If it finds a file with that name, it displays the complete path name for that file. If *filename* is not in the file system, the **find** command displays nothing when it completes.

ls

Use the **ls** (list directory) command to list the contents of a directory. The command

ls *dirname*

lists the names of the entries in the directory *dirname*. The name of the directory can be a path name. You can use **ls** command flags to get different types of information about the items in a directory. For information about the

different ways to use **ls**, see *Using the AIX Operating System* and **ls** in *AIX Operating System Commands Reference.*

pg

Use the **pg** (page) command to display the contents of a file one page at a time. The command

`pg` *filename*

displays the contents of *filename* one page at a time. At the bottom of each page on the screen, the **pg** command displays **:** (colon). To display the next screen of information, press **Enter**. For more information on the **pg** command, see *Using the AIX Operating System* or **pg** in *AIX Operating System Commands Reference.* (You also can use the **cat** [concatenate] command to display the contents of a file; however, the **cat** command scrolls through the entire file at once, rather than one page at a time.)

For more information about these commands and the different ways in which you can use them, see the appropriate entries in *AIX Operating System Commands Reference.*

Chapter 2. Routine System Management

CONTENTS

About This Chapter

This chapter explains the routine tasks associated with managing an AIX system. You may do some of the tasks described in this chapter daily, for example, starting up and shutting down the system, mounting and unmounting file systems, and backing up file systems. Other tasks you probably will do less often—setting up user accounts and creating new file systems. However, understanding the information in this chapter will help you use the AIX system effectively and keep it running smoothly.

Starting the System

To start the AIX system under normal circumstances, turn on the power switch for each system component. If there is a system diskette in drive 0 (A), the system attempts to initialize itself from that diskette. If there is no diskette in the drive, the system attempts to initialize itself from the fixed disk. Under normal circumstances, the system should initialize from the fixed disk. After the system successfully initializes from the fixed disk, it displays the copyright notice and the `login` prompt. At that time, the system is ready for normal use. For an explanation of how the initialization procedure works, see "System Initialization."

For certain system management tasks, you occasionally need to run a special version of your system: the ***maintenance system***. For an explanation of the maintenance system, see "Running the Maintenance System" on page 2-5.

System Initialization

After loading the VRM and the kernel into memory, the system goes through a period of internal initialization, during which it runs internal checks, initializes all memory, initializes some devices, and analyzes the root file system. After completing its internal initialization, the kernel starts the **init** (system initialization) process. All other processes on the system (except the scheduler) can be traced back to **init**. Most important of the processes that **init** creates are the shell and the processes that allow users to gain access to the system; that is, **init** runs the **getty** command for each ***port*** (display station) defined on the system. (For more information about the initialization processes, see **init** and **getty** in *AIX Operating System Commands Reference.*)

If the kernel is loaded from the maintenance diskette, **init** initializes the maintenance system. The maintenance system consists of only a shell with superuser authority, the **init** process, and the scheduler. (For more information about the maintenance system, see "Running the Maintenance System" on page 2-5.) To go from the maintenance system to normal operation, run **shutdown** and then press **SOFT IPL** (Ctrl+Alt+Home).

When the system is initialized for normal operation, **init** creates a shell process to run the commands in the **/etc/rc** file. The commands in **/etc/rc** prepare the system for users to log in.

To alter the way the system starts up, edit the **/etc/rc** file. Generally, **/etc/rc** contains three types of commands:

Housekeeping
: One of the first commands in **/etc/rc** is **vrmconfig**, which installs device handlers in the operating system. Several commands in **/etc/rc** make certain that the system is in proper running condition, regardless of its condition when it was last stopped. The **fsck** command, which should be the first command in

/etc/rc, checks the consistency of file systems and, as far as possible, corrects file system problems. If **fsck** cannot automatically repair the file system damage it detects, it causes the shell to stop the system. You cannot start the system until you repair the file system damage by running the maintenance system as described under "Running the Maintenance System." Other housekeeping commands remove files from temporary directories, initialize queues, save logged data, restore certain files to their original (default) states, and set up the ***ports*** that give display stations access to the system.

Mounting

The **rm -f /etc/mnttab** command in **/etc/rc** restores the mount table to its default state. Then the **/etc/mount all** command mounts all of the file systems described in **/etc/filesystems**.

Starting daemons

Daemons are processes that should start when the system starts and run until the system stops. The two most important daemons started by **/etc/rc** are **cron** and **qdaemon**.

The **cron** daemon runs the **sync** system call at 30-second intervals, completing any unfinished disk I/O. Thus, if the system fails, data on the disks cannot be more than one minute out-of-date. You can also use the **cron** daemon to start processes (specified in **/usr/lib/cron/crontab**) at preset times. For example, **cron** can start the processing of system accounting data late at night.

The **qdaemon** provides queued access to certain system resources, such as printers. For more information on queues, see "Using the Queueing System" on page 3-15.

Once it successfully completes the initialization process, **init** starts the necessary ***getty*** processes to allow display stations to use ports. After creating the initial loggers, the only function of **init** is to create loggers or set up new ports as required.

Running the Maintenance System

The AIX maintenance system provides you with a way to perform certain system management tasks without starting the system in the normal manner. You can use the maintenance system for one of two reasons:

- The system will not initialize. The maintenance system may allow you to correct the problem that prevents your system from operating normally.
- You need to perform system management tasks that would be complicated by other processes running on the system. The maintenance system is a very limited form of the operating system, consisting of just the parts of the system required to perform certain system management tasks.

The maintenance system is most useful for backing up and repairing file systems or for making any change to the system that might be complicated if other processes are running. For example, during normal startup, the operating system tries to correct any file system damage. However, some types of damage cannot be corrected automatically. In such cases, the operating system displays a message indicating that the operator must help correct the damage. To make the repairs, start the maintenance system and do one of the following:

- Use the **check a file system** maintenance command (a version of the **fsck** command described under "Checking and Repairing File Systems—The fsck Command" on page 3-6).
- Start the **standalone shell** and run **fsck** on the damaged file system.

Note: If you catch file system damage immediately, it is usually simple to repair. However, if the system runs on a damaged file system, the damage can spread, in some cases to the point that the entire file system is unusable.

You should run **fsck** only on unmounted file systems, and you should not need ***daemon*** processes while you use the maintenance system (a ***daemon*** process is a process that runs continually in the background, such as the process that manages printer queues). Therefore, when you start the maintenance system, none of the usual file systems are mounted and none of the usual daemon processes are started.

The remainder of this section explains how to start and use the maintenance system with the AIX Operating System Installation/Maintenance diskette.

To Start the Maintenance System

- If the system is not powered on:
  1. Insert the AIX Operating System Installation/Maintenance diskette into diskette drive 0 (or A).
  2. Turn on the power to all components as usual.
- If the system is powered on:
  1. Be certain that all other users log out of the system.
  2. Enter:

     ```
     shutdown
     ```
  3. When **shutdown** completes, insert the AIX Operating System Installation/Maintenance diskette into diskette drive 0 (or A).
  4. Press **SOFT IPL**.

Once initialized, the maintenance system displays the **SYSTEM MANAGEMENT** menu:

```
                         SYSTEM MANAGEMENT

 ID    Item

  1    Install the Operating System

  2    Use Maintenance Commands

  3    Start the Standalone Shell

  4    End System Management

 To SELECT an Item, type its ID and press Enter:  1
```

Select item 2 to use the maintenance commands explained under "Using Maintenance Commands" on page 2-8. Select item 3 to use the ***standalone shell*** explained under "Using the Standalone Shell" on page 2-9. To stop the maintenance system, select item 4. Then, to initialize the system for normal operation, press **SOFT IPL**.

Using Maintenance Commands

When you choose item 2 (**Use Maintenance Commands**) on the SYSTEM MANAGEMENT menu, the maintenance system displays the **USE MAINTENANCE COMMANDS** menu:

```
                    USE MAINTENANCE COMMANDS

ID    Item

 1    Show fixed disk minidisk information
 2    Change load status of a fixed disk minidisk
 3    Create a fixed disk minidisk
 4    Delete a fixed disk minidisk
 5    Check a file system
 6    Make a file system

 7    Format a diskette

 8    Restore commands
 9    Backup commands

To CANCEL and go back to the SYSTEM MANAGEMENT menu,
press F3.

To SELECT an Item, type its ID and press Enter:  1
```

To select an item, enter its number.

For more information about the function of each maintenance command, see the following:

Item ID	Reference
1-4	Information on installing the AIX Operating System in *Installing and Customizing the AIX Operating System.*
5	"Checking and Repairing File Systems—The fsck Command" on page 3-6.
6	"Creating and Mounting File Systems" on page 2-39.
7	"Formatting Diskettes" on page 2-43 and **format** in *AIX Operating System Commands Reference.*

8-9 “Backing up Files and File Systems” on page 2-45.

Note: When you choose items 8 and 9 on the **USE MAINTENANCE COMMANDS** menu to backup a file system, the backups and restores are by i-node only.

Each item leads you through a series of menus that allow you to accomplish a particular task.

Using the Standalone Shell

When you choose item 3 (**Start the Standalone Shell**) on the SYSTEM MANAGEMENT menu, the maintenance system displays the following screen:

```
#
```

Notice that the standalone shell prompt is # rather than the $ of the standard shell.

The standalone shell is most useful for tasks that you cannot perform while the operating system is running, such as repairing a file system.

When you start the standalone shell, the **init** command runs automatically, much as it does when you start the system for normal operation. Once the standalone shell is started, the following commands are available:

ar (maintains portable libraries used by the linkage editor)

backup (make backup copies of files and file systems)

chmod (change permission codes)

chown (change the owner of files or directories)

chroot (change the root directory of a command)

clri (clear i-nodes)

dd (convert and copy files)

ed (edit files)

format (format diskettes)

fsck	(check and repair file systems)
fsdb	(debug file systems)
init	(initialize system)
kill	(sends a signal to a running process)
ln	(link files)
ls	(list directory contents)
maint	(maintenance program)
mkdir	(make directories)
mkfs	(make file systems)
mknod	(create special files)
mount	(mount file systems)
od	(writes the contents of storage to standard output)
rm	(remove files)
rmdir	(remove directories)
sh	(run a shell)
sum	(displays the checksum and block count of a file)
sync	(update storage from buffers)
restore	(restore backed up files and file systems)
tctl	(send subcommands to the streaming tape device)
umount	(unmount file systems)

Most of these commands are discussed in other sections of this book (see the index) or in *AIX Operating System Commands Reference.*

File systems used during normal mode operation may be mounted from the standalone shell. The **/mnt** directory may be used for this. For example, the **root** directory of the normal system is created when the AIX Operating System is installed. The **/dev/hd0** minidisk contains this file system. To access files on the normal **root** file system from the standalone shell, use the command:

```
mount /dev/hd0 /mnt
```

Likewise, **/usr** is found on the **/dev/hd2** minidisk, **/tmp** is found on the **/dev/hd3** minidisk, and **/u** is found on the **/dev/hd1** minidisk. Thus, you could also use the command:

```
mount /dev/hd2 /mnt/usr
```

The commands of the standalone shell may be extended by mounting **/dev/hd0** and **/dev/hd2** and running commands that exist on these file systems.

1. To examine data or to run normal mode commands, enter:

   ```
   mount /dev/hd0 /mnt
   mount /dev/hd2 /mnt/usr
   mount /dev/hd1 /mnt/u
   mount /dev/hd3 /mnt/tmp
   PATH=/mnt/bin:/mnt/usr/bin:$PATH
   cd /mnt
   ```

2. To check the **root** file system, enter:

   ```
   fsck /dev/hd0
   ```

3. To check the **/usr** file system, enter:

   ```
   fsck /dev/hd2
   ```

4. To make backups by name, enter:

   ```
   mount /dev/hd0 /mnt
   mount /dev/hd2 /mnt/usr
   mount /dev/hd1 /mnt/u
   mount /dev/hd3 /mnt/tmp
   backup -i
   cd /mnt
   ```

   (Enter the file names relative to the current working directory, that is, **etc/system**, **bin/cp**.)

To end the standalone shell, press **END OF FILE**.

Stopping the System

The AIX system can run multiple processes, including background processes, at the same time. To prevent damage to the file system, you should bring all processes to an orderly stop before you turn off the power to the computer. The **shutdown** command (described from a user's perspective under "Starting the System" on page 2-4) is the usual way to bring the system to an orderly stop.

During the default shutdown, all users are notified through the **wall** command of the impending system shutdown. The **hold** command prevents any new logins. After the specified number of *seconds* (60 by default), the system stops the accounting and error-logging processes. **shutdown** then runs the **killall** command to end any remaining processes and runs the **sync** command to flush all memory resident disk blocks. Finally, it unmounts the file systems and sends the appropriate signal to **init** and prints the message: `....Shutdown completed.....` At this point, you can safely turn off the power switch on the system unit.

Note: If you have a single user system, you may wish to do a `shutdown -f` to eliminate the message from the **wall** command and the delay.

For information related to stopping the system, see "Disk Buffering—the sync Command" on page 3-4.

Managing User Accounts

Giving users access to the system is an important system management task. Even if you are the only user on the system, you may need to create accounts, remove accounts, and change account information occasionally. You can perform these account management tasks with the **users** command described under "Creating, Changing, and Removing Accounts—The users Command." To manage the accounts on your system most effectively, you also should be familiar with the information under "Types of User Accounts" on page 2-23, "Using Different Log In Names" on page 2-25, and "User Account Files" on page 2-26.

Note: Another name for **users** is **adduser**.

Creating, Changing, and Removing Accounts—The users Command

The **users** command is a tool for managing user accounts. **users** provides the following subcommands:

add	Creates new user accounts and new groups.
change	Changes account and group information, such as user name, password, and group ID.
delete	Removes accounts or groups from the system.
help	Displays the **users** subcommands.
invalidate	Makes a user password invalid.
quit	Puts changes into effect and ends **users**.
show	Displays user or group account information.

To run the **users** command, you must be a member of the system group or have superuser authority.

Starting and Stopping users

To Start users

- Enter `users`.

After you start **users**, you can use any of its subcommands to perform your account management tasks. Enter ? (question mark) to display the list of subcommands:

```
Available commands are:

a[dd]          user or group
c[hange]       user or group
d[elete]       user or group
h[elp]         print this message
i[nvalidate]   a user
q[uit]         finalize changes and exit
s[how]         user or group

<DEL> will exit without changing any of the files.
>
```

Note: The DEL key referred to in this message corresponds to **INTERRUPT**.

To select a subcommand, type its initial letter (for example, `a` for **add**) and press **Enter**. If the subcommand requires you to specify `user` or `group`, use the abbreviation `u` or `g` (for example, `a u tom`).

To show the information about a user, enter: `show user` *username* (or `s u` *username*). To show the information about a group, enter: `show group` *groupname* (or `s g` *groupname*).

To display help information, enter: `h`.

To Stop users

- To stop **users** without making any changes, press **INTERRUPT**.

OR

- To put changes you have made into effect and stop **users**, enter `q`.

The following sections explain how to use the remaining **users** subcommands (**add**, **change**, **delete**, and **invalidate**).

Using the a[dd] Subcommand

To Add a User

1. Enter: `users` (If **users** is already started, go to step 2.)
2. After the `>` prompt, enter: `a u` *username*. (Use lowercase letters for the names of users you add to the system.)
3. To add user information, enter: `n` after `OK?  (y)`.
4. To add the user's full name:

   a. Enter: `fu` after the `Field?` prompt.
   b. Enter the user's full name (for example, john doe).
5. To add a password:

   a. Enter: `pa` after the `Field?` prompt.
   b. Enter the password.
6. To add other information:

   a. After the `Field?` prompt, enter enough of the field name to identify it.
   b. Enter the information.
7. When your changes are complete, press **Enter** after the `Field?` prompt. **users** displays the new information (passwords are encrypted).
8. If the changes are correct, press **Enter** after the `OK?  (y)` prompt to add the new user to the system. (If the changes are not correct, enter `n` and then make the necessary corrections).
9. Press **Enter** after the prompt `Standard new user initialization?  (y)`.
10. Enter another subcommand or stop **users**.

Note: You are not allowed to change the information in the **opt. groups** field from the **add user** subcommand. To add a user to a group, use the **change group** subcommand.

You can set the values for other items with the same procedure used to set `Fullname` and `Password`. For more information about user accounts, see "Types of User Accounts" on page 2-23.

To Add a Group

1. Enter: users. (If **users** is already started, go to step 2.)
2. After the > prompt, enter a g *groupname*.
3. To accept the displayed information, press **Enter** after the OK? (y) prompt. (To add group information, enter n after the OK? (y) prompt.)
4. To add a group password:
   a. Enter pa after the Field? prompt.
   b. Enter the group password.
5. To add members to a group:
   a. Enter m after the Field? prompt.
   b. Enter a.
   c. Enter the username of the user you are adding.

   Note: Be sure to type the name correctly because there is no check to verify if this name is defined.
6. To end this procedure, press **Enter** after the Field? prompt.
7. If the changes are correct, press **Enter** after the OK? (y) prompt to add the group to the system.
8. Enter another subcommand or stop **users**.

For more information about groups and the fields used by the **add group** subcommand, see "The Group File" on page 2-30.

Using the c[hange] Subcommand

To Change User Information

1. Enter: `users`. (If **users** is already started, go to step 2.)
2. After the `>` prompt, enter `c u` *username*.
3. To make changes to the displayed information, enter `n` after the `OK? (y)` prompt.
4. Enter the name of the field you want to change after the `Field?` prompt.
5. Enter the information for the field you specified.
6. Use the same method to change the information in other fields.
7. When your changes are complete, press **Enter** after the `Field?` prompt.
8. If the changes are correct, press **Enter** after the `OK? (y)` prompt. **users** displays the `[UPDATED]` message.
9. Enter another subcommand or stop **users**.

For more information about the fields for which you can change values, see "The /etc/passwd File" on page 2-27.

Note: You cannot change the information in the **opt. groups** field from the **c u** subcommand. To add a user to a group, use the **change group** subcommand.

To Change Group Information

1. Enter: `users`. (If **users** is already started, go to step 2.)
2. After the `>` prompt, enter `c g` *groupname*.
3. To make changes to the displayed information, enter `n` after the `OK? (y)` prompt.
4. Enter the name of the field you want to change after the `Field?` prompt. (For example, to change the members shown as part of the group, enter `m` or `member`.
5. Enter `a` or `d`.
6. Enter the user name you wish to add or delete. Be sure to type the name correctly because there is no check to verify if this name is defined.
7. When your changes are complete, press **Enter** after the `Field?` prompt.
8. If the changes are correct, press **Enter** after the `OK? (y)` prompt. **users** displays the `[UPDATED]` message:
9. Enter another subcommand or stop **users**.

Using the d[elete] Subcommand

To Delete a User

1. Determine if you want to save the files in the user's directory. If you do, change the name of the directory, change the owner of the directory, or follow the instructions in step 3 for saving files. If you do not want to save the files, remove all files (including the hidden files such as the **.profile** file) from the directory of the user that is to be deleted.

2. Enter: `users`. (If **users** is already started, go to the next step.)

3. After the `>` prompt, enter `d u` *username*. **users** displays the prompt `Should the login directory (/u/`*username*`) be removed? (y)`. If you want to save the files in that directory, type `n` and press the **Enter** key.

   If you have already removed all the files in that directory, press the **Enter** key, and you should see the `[DELETED]` message.
   (If you have not removed all files from the user's login directory, **users** displays the message, `The user cannot be deleted because the user's login directory is not empty`.

4. To verify that the user is deleted, enter `s u` *username*. **users** should display a message indicating that there is no such user.

5. Enter another subcommand or stop **users**.

Notes:

1. Be sure to change the ownership of (or delete) all files owned by the user who is being deleted, including files which may reside in directories other than the **/u/*username*** directory. Failure to do so may cause problems later. The owner of a file is identified by the user number (UID). Later, when a new user is added to the system, the same number may be used by the system again. The new user can then own any of the files which were not removed or for which ownership was not changed. For information about changing the ownership of a file, see the **chown** command in *AIX Operating System Commands Reference.*

2. When you delete a user, you have the choice of removing all of the files in that user's login directory or answering `no` to the prompt, `Should the user's login directory be removed?` If you answer `no` to the prompt, the entry for `username` is removed from the **/etc/passwd** file, but the **/u/username** directory will not be removed. You would not have to copy the files you want to save to another directory, but you would need to change the ownership of the files in the **/u/username** directory and of any other files owned by that user.

To Delete a Group

1. Enter: `users`. (If **users** is already started, go to the next step.)
2. After the `>` prompt, enter `s g` *groupname*. Write down the names of all the users listed under `Members`.
3. After the `>` prompt, enter `s u` *username* for one of the members of the group. Check to see if the group you want to delete is listed under `Group`. If it is, use the **c u *username*** subcommand to change the **Group** field. Repeat this step for each user you listed in the previous step.

   Note: See the example below for an alternate method for determining which users use the group you want to delete as their primary group. If there is a large number of members in the group, you may prefer the alternate method.
4. After the `>` prompt, enter `d g` *groupname*. **users** displays the `[DELETED]` message.
5. To verify that the group is deleted, enter `s g` *groupname*. **users** should display a message indicating that there is no such group.
6. Enter another subcommand or stop **users**.

Note: Before deleting a group, change the group field for all users using that group as their primary group. The next group added is assigned that group **id**. Users may get incorrect group assignments.

You can use an alternate method to delete a group. In the following example, the group is named **group1**. Use information in the **/etc/group** and **/etc/passwd** files to determine which users have **group1** as their primary group.

Enter `pg /etc/group` to display the group file. For more information about the **/etc/group** files see “The Group File” on page 2-30. The following example shows **group1** with a group number (GID) of 200:

```
system::0:root,su
staff::1:user2,user3,user4,user5
bin::2:root,su,bin
sys::3:root,su,bin,sys
adm::4:root,su,bin,adm
mail::6:root,su
usr::100:guest
group1::200:user3,user4,user1
```

Note that the group number for group1 is 200. You can also get this information by using the **s g group1** subcommand after starting **users**. The group number is shown in the GID field.

Enter `pg /etc/passwd` to display the password file. For more information about the **/etc/passwd** file, see “The /etc/passwd File” on page 2-27. The following example shows **group1** with a group number (GID) of **200** for the primary group:

```
root:!:0:0::/:
su:!:0:0::/:
daemon:!:1:1::/etc:
netmail:!:1:1::/usr/spool/qftp:/usr/lib/INnet/waxsrvr
bin:!:2:2::/bin:
sys:!:3:3::/usr/sys:
adm:!:4:4::/usr/adm:
uucp:!:5:1::/usr/spool/uucppublic:/usr/lib/uucp/uucico
adduser:!:0:0::/usr/adm:/etc/adduser
guest:!:100:100::/usr/guest:
name:!:201:0::/u/name:
user1:!:200:200::/u/user1:/bin/sh
user2:!:202:1::/u/user2:/bin/sh
user3:!:1::/u/user2:/bin/sh
user4:!:202:1::/u/user2:/bin/sh
user5:!:202:1::/u/user2:/bin/sh
```

Notice in the above example, that **user1** is the only entry with **200** as the group number field. It is the only entry in the above example with **group1** as the primary group. Using the information from the **/etc/group** and the **/etc/passwd** files, you can change the primary group for **user1** by using the **users** command as described below:

1. Enter: `users`.
2. After the `>` prompt, enter `c u user1`.
3. To change the displayed information, enter `n` after the `OK? (y)` prompt.
4. To change the primary group:
   a. Enter: `gr` after the `Field` prompt.
   b. Enter the name for the new primary group.
5. Press Enter after the `Field?` prompt.
6. If the changes are correct, press Enter after the `OK?` prompt. **users** displays the `[UPDATED]` message:
7. After the `>` prompt, enter `d g group1` to delete the group. **users** displays the `[DELETED]` message.

8. To verify that the group is deleted, enter `s g group1`. **users** should display a message indicating that there is no such group.

9. Enter `q` to stop **users**.

Note: When deleting a group, check and see if any users have it as their primary group. If they do, change the primary group for those users. Failure to do so may cause problems later. The primary group for a user is identified by the group number (GID). Later, when a new group is added to the system, it may be assigned the same GID that was used by the deleted group. Users whose primary group was not changed would then have the new group as their primary group.

Files also need to be checked. It may not be practical to change the group ownership of all files created by a deleted group. However, there may be particular files which you may want to reassign. Use the **chgrp** command to change the group ownership of a file. For more information about the **chgrp** command, see *AIX Operating System Commands Reference*.

Because of the possible problems involved in using the **delete group** subcommand, use it sparingly. Give careful consideration to the consequences before deleting a group.

Invalidating and Reinstating Users

You can prevent a user from logging in to the system by invalidating the user. You do not have to delete a user's files to invalidate the user. There are two ways to invalidate the user. Both involve the **users** command.

To Invalidate a User

1. Enter: `users`. (If **users** is already started, go to step 2.)

2. After the `>` prompt, enter `i` *username*. **users** displays the `Invalidating` *username* message.

   Note: Do not specify `u` with the **i** subcommand.

3. Enter another subcommand or stop **users**.

To reinstate the user, use the **(c)**hange subcommand to clear the user's program field.

You can also use the **restrictions** attribute in the **/etc/security/passwd** to invalidate the user. Set this attribute to **nouse** to invalidate a user. To reinstate a user, set this restriction to **none**. For more information about **/etc/security/password** and these restrictions, see Chapter 6, "Managing System Security" on page 6-1.

Types of User Accounts

Each entry in the **/etc/passwd** file identifies an account. ("The /etc/passwd File" on page 2-27 describes the **/etc/passwd** file.)

There are two types of accounts on the AIX system:

- The account with superuser authority
- User accounts.

If you know the user name and password associated with an account, you can log in as that user. Thus, you can have one or more user accounts for your regular work and also, when necessary, use the account with superuser authority for system management work.

root—The Account with Superuser Authority

The user name ***root*** identifies the one user who operates without any restrictions—the user who has ***superuser authority***. (Another name for root is ***su***, for superuser.)

When you have superuser authority, you are immune from all permission checks in the system, you have read, write, and execute permission for every file in the system, you can issue any system call, and you can cancel any process.

Because none of the usual protections apply when you have superuser authority, you should be extremely careful when you are logged in as root. Make it a practice never to log in as root when you can log in as a regular user and accomplish the same task. On a system that serves several different people, it is good practice to strictly limit who knows the root password, and to change the password often, in order to reduce the chance of accidental damage.

Note: If a user account has a password, you can also use the root password to log in to it.

You can obtain superuser authority in three different ways:

- After you log in with your normal user name, enter the **su** (switch user) command and supply the root password when prompted to do so. The shell prompt changes (usually from `$` to `#`) to remind you that you have superuser authority. To return to normal user status, press **END OF FILE**. This is usually the most convenient way to obtain and give up superuser authority.
- Log in with the one of the user names `su` or `root`. Instead of the usual `$` (shell) prompt, the system displays the root prompt (usually `#`), to remind you that you have superuser authority. After you finish the work that requires superuser authority, either log out (press **END OF FILE** and the `$` prompt returns) or run the **shutdown** command to stop the system.
- Start the maintenance system (see "Running the Maintenance System" on page 2-5). The maintenance system shell has superuser authority.

User Accounts

All user accounts are subject to normal system checks and protections. To reduce the chance of accidental damage to the system, always work with a user account except when you must have superuser authority.

There are two types of user accounts: those created with the **users** program (described under "Creating, Changing, and Removing Accounts—The users Command" on page 2-13) and those supplied with the system to perform certain system management tasks. Accounts created with **users** can be called ***ordinary user accounts***. These accounts are commonly identified with a particular person. They are used to run application programs (for example, the shell, text editors, or electronic spreadsheets) and store the data associated with them. As with any account, there is an entry in the **/etc/passwd** file and in the **etc/security/passwd** file for each ordinary user account. The home (or log in) directories for ordinary user accounts are located in the **/u** directory (for example, `/u/jones`).

The second group of user accounts is supplied with the system. They give users access to certain data and the ability to perform certain system management tasks which otherwise would require superuser authority.

Following is a list of these special user accounts:

bin
: The user **bin** owns most of the directories, libraries, and programs or commands provided with the system. While most users have **x** (execute) permission for the files owned by **bin**, only **bin** can modify those files. Normally, you should **su** to **bin** to install new programs and libraries.

  In most respects, **bin** has ordinary user privileges. However, because **bin** owns the very important directories **/bin**, **/usr/bin**, **/lib**, and **/etc**, any mistakes you make when running as **bin** could affect the entire system.

  bin also owns the directory that contains the special files for system disks, which are write- and read- protected from other system users. When you run a command that must read system disks (for example, **df**), the system sets the user name associated with that command to **bin**. (Such commands are called ***suid*** [set user id] programs; the **suid** feature can work with any user name, including root, depending upon the requirements of the program.)

users
: The user **users** (or **adduser**) is limited to one function: creating and maintaining user accounts. After logging in, **users** runs the account maintenance program **users** (which is described in detail under "Creating, Changing, and Removing Accounts—The users Command" on page 2-13).

adm
: The user **adm** owns the administrative and accounting files in **/usr/adm**. The **adm** user allows you to isolate billing from other system management tasks.

In addition to these special user accounts, there is also a special user group—the system group, or group **0**. Certain commands (in the directory **/etc**) which are not available to regular users can be run by any member of the system group.

Two good reasons for having a system group are to:

- Limit the number of users who need to know the root password. Members of the system group can do many system management jobs without obtaining superuser authority. However, because members of the system group do not have superuser authority, any mistakes they make are less likely to cause serious problems.
- Increase the level of security provided to ordinary system users. Superuser authority gives a user complete access to every file on the system. Members of the system group do not have such access. Thus, members of the system group can perform system management tasks with less chance of compromising the security of data stored on the system.

Using Different Log In Names

Once you are logged in to the system, the most convenient way to change the account name you are using is with the **su** (switch user) command. To obtain superuser authority with the **su** command, simply enter **su** and then enter the superuser password when the system prompts you for it:

```
$ su
password:
# _
```

Notice that when you obtain superuser authority, the shell prompt changes from $ to #. To return to your original user name, press **END OF FILE**.

You also can use **su** to change your user name to that of any other user on the system, provided you know the required password:

```
$ su username
password:
$ _
```

Notice that you must supply the **su** command with *username* any time except when you are obtaining superuser authority. To return to your original user name, press **END OF FILE**.

For more information about the **su** command, see **su** in *AIX Operating System Commands Reference*.

When you use a number of different accounts, and especially if you often change from one account to another with the **su** command, you may forget either your current user name or the user name you used when you logged in to the system. Use the **id** command to determine your current user name:

```
# id
uid=0(root) gid=0(system)
#
```

Regardless of your current user name, you can use the **logname** command to determine the user name you used to log in to the system:

```
$ logname
jones
$
```

You can also use the **who am i** command to determine your user name:

```
$ who am i
jones     console     Apr 4  15:04
$
```

In addition to your user name, the **who am i** command also gives you the name of the display station you are using (**console** in this example) and the date and time that you logged in.

User Account Files

To determine the authority associated with a user account, the system refers to four files:

/etc/passwd

/etc/security/passwd

/etc/group

/etc/security/group

The next two sections explain the information these files contain and how you can modify these files as necessary to maintain user accounts on your system.

The /etc/passwd File

The **/etc/passwd** file contains all information that defines a user and contains one line for each user on the system. Each line contains seven fields separated by **:** (colons), for example:

```
bud:!:200:1:Bud Smith/3000;tech:/u/bud:/bin/sh
 1  2  3  4          5              6       7
```

A5ACG012

Figure 2-1. The Fields in a Password File Entry

The fields, from left to right, are:

1. User name
2. Encrypted password (contains a ! (exclamation point) as a placeholder)
3. User number **(UID)**
4. Group number **(GID)**
5. Optional information field
6. Home (login) directory
7. Initial program (***login shell***).

Following are descriptions of the information in each field:

User name
: The **user name** field contains the name (up to eight characters long) with which a particular user logs in to the system. The system always uses the name in this field to refer to that user.

Encrypted password
: This field contains a placeholder (!). The encrypted password is stored in the **/etc/security/password** file.

User number and **Group number**
: The user number **(UID)** and group number **(GID)** define the identity of users for security purposes. All file protection on the AIX system is based on three sets of permissions, those for **owner**, **group**, and **others**. Every process started by a user is identified with that user's user number and group number. Thus, the user number and group number together with permissions make it possible to control the files to which a particular user has access.

Optional information

The fifth field of the password file contains assorted information related to the user. It consists of three optional subfields in the form:

fullname/filesize;siteinfo

fullname The real name of the person whose user name appears in this field.

filesize This subfield specifies the maximum size (in blocks) of files that the user is allowed to create. If you do not set the maximum file size, you can omit the separator (/), and a standard, system-wide value will apply to that user's files. (For more information on setting maximum file size, see **ulimit**, described under **sh**, and **login** in *AIX Operating System Commands Reference.*)

siteinfo This subfield can contain whatever information you want to include. If you do not include the *siteinfo* field, you can omit the separator (**;**).

Home (login) directory

The sixth field contains the name of the user's home (login) directory. The **login** program places the user in this directory at login. Typically, this directory is owned by the user and all the user's private files are kept in the directory tree below it. When you add a new user to the system, the **users** command automatically creates a login directory for the user.

Initial program (log in shell)

The last field in the password file contains the name of the program that the user runs upon logging in (the login shell). Usually this is the **/bin/sh**, or shell program (described in *Using the AIX Operating System* and under **sh** in *AIX Operating System Commands Reference*). If this field is blank, the system automatically starts the **/bin/sh** program for that user. You can specify another program in this field if, for example, the user requires a different command interpreter.

Figure 2-2 on page 2-29 shows a typical password file. The entries for certain users (for example, **root** and **bin**) appear on all systems, but with different encrypted passwords.

This sample password file includes users in different groups and users with special login programs:

```
root:!:0:0::/:
users:!:0:1::/usr/adm:/etc/adduser
daemon:!:1:1::/:
bin:!:2:2:/3000:/bin:
adm:!:4:4::/usr/adm:
sync:!:20:1::!:/bin/sync
chris:!:201:1:Chris Cooper;tech:/u/chris:
heinz:!:202:1:Heinz Hart;mgmt:/u/heinz:
ted:!:203:1:Ted Black;tech:/u/ted:
jim:!:204:1:Jim Mori;tech:/u/jim:
bud:!:200:1:Bud Smith/3000;tech:/u/bud:/bin/sh
```

Figure 2-2. Sample /etc/passwd File

It is good practice to reserve some number of user numbers (50 to 200) for use by programs that are created on your system and require special privileges. The remaining user numbers then are available for ordinary users.

The /etc/security/passwd File

Security-relevant information about user passwords is stored in the **/etc/security/passwd** file. Entries can be modified only by a user with superuser authority. Modify **/etc/security/passwd** in the same way as **/etc/passwd**.

The following fields are contained in this file:

password A character string that authenticates the user. This attribute is set to the encrypted value of the password when you add an account. A null password indicates that a password is not needed to log in to this account.

lastupdate The date on which the password was last changed. Normally, the **passwd** program sets this value. Set it to 0 to force an immediate password change when an account is added.

restrictions
An attribute that determines how and whether this account can be used. It should contain either **none, nouse,** or **nologin**.

Note: If one of the above is not specified, the default is **none**.

auditclasses
An attribute that defines the audit events for which a user is to be audited. See "Example Audit Class Configuration" on page 6-24.

The Group File

You can assign a user to one or more groups. Each group shares certain protection privileges. For example, you may want to place users in the same group because they work on the same project and need access to a common set of files. Or you may create a group (like the system group) to give certain users special privileges. Thus, you can use groups to increase or restrict the privileges of the users assigned to them.

When a user logs in, the system assigns the user to the group specified by the group number in that user's entry in **/etc/passwd**. The user can then use the **newgrp** command (described in *AIX Operating System Commands Reference*) to change the group associated with all processes the user creates. The group under which the user is currently working is called the primary group.

The AIX system allows users to belong to several groups at the same time. A user's primary group is the one specified in the **/etc/passwd** file. However, you can use the **users** command (described under "Creating, Changing, and Removing Accounts—The users Command" on page 2-13) to add users to other groups. (Also use **users** to create new groups.) Any files created by a user belong to that user's primary group, but the user has access to all files accessible to any group to which he belongs. To change the primary group, use the **newgrp** command (described under **newgrp** in *AIX Operating System Commands Reference*).

The file **/etc/group** defines which users make up each group. Each line in **/etc/group** defines a group and consists of four fields separated by colons:

1. Group name
2. Encrypted password (empty field; if an encrypted password exists in **/etc/security/group**, this field contains the placeholder **!** (exclamation point)).
3. Group number (GID)
4. Permission list (user names separated by commas).

The **group name** is a string of up to eight characters used to refer to the group. If there is a **password**, any user who attempts to enter the group is required to use it. The **group number** is simply the number assigned to that group. The **permission list** contains all users who have permission to enter the group, for example:

```
system::0:root,bin,jim,steve
staff::1:
managers::200:heinz,ted
```

The **users** command usually puts new users in the **staff** group. You can add users to groups and create new groups as necessary. It may be useful to establish a group for new users; such a group, often called **other**, allows new users to work on the system before they are assigned to a particular group. As with user numbers, it is good practice to reserve some group numbers for uses unique to your system (perhaps 50 to 200 in all, with the first 10 to 20 reserved for use in system management tasks). The remaining group numbers can be assigned to users.

For users to change their primary group, they either must be on the list of users permitted to enter that group or, if the group has a password, know the password for that group. The one exception to this rule is that a user always can return to the group specified in the password file (the primary group). That is, a user does not need to give a password or appear in the permission list in order to return to their primary group.

Note: Group passwords are rarely necessary. In practice, the permission list usually provides adequate group security, and most systems do not use group passwords.

The /etc/security/group File

This file contains information important to the security of group passwords. It is modified the same way as **/etc/group**, except that it can only be modified by a user with superuser authority.

password A character string that authenticates the group. Set this attribute when you add an account. A null password indicates that a password is not needed to log in to this account.

Tailoring the User Environment

Certain variable factors control how the shell operates. The way these variables are set determines the shell's ***environment***. Several files control the environment and thereby control many of the programs that users run. While the AIX system includes prototypes for these files, you may need to change the prototypes, or add to them, to adapt the system to your needs.

This section will be easier to understand if you are somewhat familiar with the AIX shell. For information about the AIX shell, see *Using the AIX Operating System*.

The shell provides named variables and mechanisms for assigning string values to them, testing them, and substituting them into commands. In addition to their use as program variables for the shell as a high-level programming language, some of these variables control how the shell works. Furthermore, **exported** shell variables are passed in the process environment from the shell to the programs that it runs as user commands. Since exported variables can affect how any program runs, they provide potentially wide-ranging control over the user environment. Some of the commonly used environment variables are discussed under **sh** in *AIX Operating System Commands Reference*; others are discussed in the same book in conjunction with specific commands. This section discusses where the shell obtains exported variables other than those explicitly exported by the user.

/etc/environment

Before any user logs in to the system, the **init** process (the main process involved in starting the system) reads the file **/etc/environment**. The **init** process passes variable assignments made in **/etc/environment** to each process it creates (its ***child*** processes). These variables automatically become part of the list of exported shell variables. To modify the **/etc/environment** settings, use a text editor to change the **/etc/environment** file.

Note: **/etc/environment** can contain only variable assignments, not shell commands.

Unless the values shipped with your system are correct, you should set at least the following variables in **/etc/environment**:

TZ This **TZ** (time zone) variable controls how your system translates its standard time (Universal Coordinated Time, also known as Greenwich Mean Time) into local time. The form of the TZ variable assignment is:

`TZ=`*nnnhddd*

- *nnn* is the three-character name of your normal time zone.
- *h* is the number of hours by which your time zone normally lags behind standard system time. For locations east of Greenwich, you can use the

form *-h* to specify the number of hours by which your time zone leads standard system time.

- *ddd* is the three-character name for your time zone during daylight savings time. (Omit this field if daylight savings time is not observed locally.)

Some typical TZ assignments are:

TZ = EST5EDT	**E**astern **S**tandard **T**ime, **5** hours behind Greenwich, **E**astern **D**aylight **T**ime.
TZ = CST6CDT	**C**entral **S**tandard **T**ime, **6** hours behind Greenwich, **C**entral **D**aylight **T**ime.
TZ = MST7MDT	**M**ountain **S**tandard **T**ime, **7** hours behind Greenwich, **M**ountain **D**aylight **T**ime.
TZ = PST8PDT	**P**acific **S**tandard **T**ime, **8** hours behind Greenwich, **P**acific **D**aylight **T**ime.
TZ = GMT0	**G**reenwich **M**ean **T**ime, **0** hours behind Greenwich. Note that, as this example shows, when daylight savings time is not observed locally, the *ddd* field must be omitted.
TZ = CET-1	**C**entral **E**uropean **T**ime, **1** hour before Greenwich. Note that *h* is specified hours before Greenwich and, again, the *ddd* field is omitted.

PATH Executable files (programs or commands) are located in various directories. The **PATH** variable contains a list of directory names (separated by colons) that the shell and other commands use when searching for an executable file. In the PATH assignment, a **:** (colon) that is not preceded by a directory name stands for the current directory. If the PATH variable is not otherwise set, its default assignment is:

```
PATH=/bin:/usr/bin:/etc::
```

With this assignment, a command searches the following the directory **/bin**, the directory **/etc**, the directory **/usr/bin**, and finally the current directory(:).

Two common reasons for setting the PATH variable to something other than its default are:

- To increase the efficiency of searches. In some cases, a PATH that searches the current directory last is most efficient. The following assignment causes the current directory to be searched last: **PATH = /bin:/usr/bin:/etc::**
- To include other directories in the search. For example, if you have a directory named **/local** containing commands that you have created, you use the assignment **PATH = /bin:/usr/bin:/local:/etc::** to include it in the search path. If you use this assignment, the **/local** directory will be searched before both the **/etc** and current directory.

filesize The **filesize** variable sets a default value for the system-wide maximum file size (in 512-byte blocks). The default file size limit is 2048 blocks or slightly more than one million bytes. You can override the default setting for individual users by setting a different value for **filesize** in the **/etc/passwd** file. (For more information on setting maximum file size, see **ulimit** described under **sh** and **login** in *AIX Operating System Commands Reference.*)

Login

The **login** program adds the following values to the environment:

LOGNAME (user name)

TERM (display station, or **terminal**, type from **/etc/ports**)

HOME (login directory).

/etc/profile and $HOME/.profile

An initial user shell includes in its environment the variables passed to it by **login**. Then, before the shell accepts any input from the user, it runs the commands in **/etc/profile** and in the file **.profile** in the user's login directory. Thus, **/etc/profile** should contain any commands that all users need to run when they log in (for example, the **news -n** command, which informs users of items in the system news facility). At the same time, **/etc/profile** should not contain nonessential commands. (Although the system runs all commands in **/etc/profile**, individual users can override the variable assignments made in **/etc/profile** with assignments in their private profile files, **$HOME/.profile**.)

Generally, the **umask** command runs from **/etc/profile**. A ***mask*** determines the permissions that AIX sets initially for a file or directory. The **umask** command creates this mask. AIX applies the mask to the permission code **777** for directories and **666** for files. As a parameter, the **umask** command takes a string of three octal digits; the digits specify which permissions to remove. For example, the default **umask** command, which removes write permission for group and others, is:

```
umask 022
```

This mask causes AIX to create directories with the permission code **755** and files with the permission code **644**.

You also can run **umask** from the command line or from your **$HOME/.profile**. In either case, you set the file creation mask for your own processes only, not for all system users.

(For a general discussion of permissions, including octal representation, see *Using the AIX Operating System.*)

Two environment variables should be set in **/etc/profile** and marked for export: **MAIL** and **MAILMSG**. The shell checks the last modification date of the file **MAIL** to determine whether the user should be notified of the receipt of new mail, and **MAILMSG** is the notification message itself. If you use the MAIL facility on your system, the following variable assignments should appear in **/etc/profile**:

```
MAIL=/usr/mail/$LOGNAME
MAILMSG="[You have new mail]"
export MAIL MAILMSG
```

You also may set the **TIMEOUT** variable in **/etc/profile**. The value of **TIMEOUT** is specified in minutes. When a user remains at command level for *TIMEOUT* minutes without entering a command, the shell exits and logs off the user. Generally, the **TIMEOUT** variable is most useful on systems that serve a number of users or on systems where security is a major concern (where, for example, a display station left unattended for a long time could give unauthorized people access to classified data).

If there are commands that you believe most users will want to run when they log in, but that are not appropriate for **/etc/rc**, you can create a default **.profile** file to be added automatically to the home directories of new users. The **users** command and the autolog function makes it possible to set up default **.profile** files through the **/usr/adm/newuser.usr** command file, which runs when a user is added to the system. This file copies **def.profile** from **/usr/adm** to the **$HOME** directory of the new user. For more information on **users**, see "Managing User Accounts" on page 2-13.

Information about File Systems—The /etc/filesystems File

Information about file systems is stored in the file **/etc/filesystems**. Most of the file system maintenance commands use **/etc/filesystems** to relate file system names to corresponding devices and to provide information about file systems to those commands.

The **/etc/filesystems** file is organized into ***stanzas*** that describe file systems. A stanza has the same name as the file system that it describes, and contains a series of *attribute* = *value* pairs that specify the characteristics of the file system. Stanzas named **default** contain information common to several file systems.

Following is part of a sample **/etc/filesystems** file:

```
*   This file describes all of the known file systems.  It
*   is used by most of the file system maintenance routines
*   to map file system names into the corresponding device
*   names.  It also provides the information which tells fsck,
*   df, mount and other programs what file systems to operate
*   on by default.

default:
        vol       =  "AIX"
        mount     =  false
        check     =  false
        free      =  false
        backupdev =  /dev/rfd0
        backuplen =  2400

/:
        dev       =  /dev/hd0
        vol       =  "root"
        mount     =  automatic
        check     =  false
        free      =  true
```

Stanzas can contain attributes in addition to the ones shown in the example. Following is a list of the possible **/etc/filesystems** attributes and a brief explanation of their meanings:

account Determines file system to be processed by the accounting system. The value can be either **true** or **false**.

backupdev Used by the **backup** and **restore** commands to determine the backup device associated with each file system. The value is usually the name of a diskette or magnetic tape special file.

backuplen Used by the **backup** command to determine the size of the default backup device associated with each file system.

backuplev Used by the **backup** command to determine the default backup level for each file system. For an explanation of backup levels, see "Backing up Files and File Systems" on page 2-45.

boot Used by the **mkfs** command to initialize the boot block of a new file system. The value of **boot** specifies the name of the load module to be placed into the first block of the file system.

check Used by the **fsck** command to determine the file systems to be checked. The value can be **true**, **false**, or some number that corresponds to a particular phase of the **fsck** program. For more information about **fsck**, see "Checking and Repairing File Systems—The fsck Command" on page 3-6.

cyl Used by the **mkfs** command to initialize the free list and superblock of a new file system. The value is set the number of blocks in one cylinder.

dev Identifies the block special file where the file system resides. System management programs use this attribute to relate file system names to the appropriate device names.

free Used by the **df** command to determine which file systems are to have their free space displayed. This value is set either **true** or **false**.

mour Used by the **mount** command to determine whether to mount the file system. If the value of **mount** is **true**, then the **mount all** command mounts the file system. The most convenient way to accomplish routine file system mounting is to place the **mount all** command in the **/etc/rc** file. The value of mount can be set **true**, **false**, or **readonly**. When the **readonly** value is set, the file system is mounted, but its contents cannot be changed.

The **mount** attribute for the root file system has a special value: **automatic**. The **automatic** value causes the root file system to be mounted whenever the system is initialized and prevents the **mount all** command from attempting to mount the already-mounted root file system.

size Used by the **mkfs** command to specify the size of a new file system. The value of **size** is some number of 512-byte blocks.

skip Used by the **mkfs** command to initialize the free list and superblock of a new file system. The value of **skip** is some number of 512-byte blocks to be skipped between data blocks (the ***interleave factor***).

vcheck Used by the **varyon** command to determine the file systems to be checked. **varyon** calls the **fsck** command, which uses **vcheck** in the same way that it uses **check**. See the description of the **check** attribute in this section. For more information about **varyon**, see "The varyon Command" on page 4-39.

vmount Used by the **varyon** command to determine whether to mount the file system. **varyon** calls the **mount** command, which uses **vmount** in the same way that it uses **mount**. See the description of the mount attribute in this section. For more information about **varyon**, see "The varyon Command" on page 4-39.

vol Used by the **mkfs** command when it initializes the label on a new file system. The value is a volume label with a maximum length of six characters.

Creating and Mounting File Systems

A file system is a complete directory structure confined to a single minidisk or diskette. Before you can use a minidisk or diskette to store data, create a file system on it. Use the **mkfs** (make file system) command to create file systems.

Note: The **mkfs** command is not the only way to create a file system on a minidisk. For more information on creating file systems, see *Installing and Customizing the AIX Operating System*.

The general form of the **mkfs** command is:

`/etc/mkfs` *name size*

where *name* is the name of the minidisk on which the file system is to be created and *size* is the total size of the file system in 512-byte blocks. Allocate more space than you need because the i-nodes use up to 5% of the blocks.

Warning: The **mkfs** command will erase all information on the device you specify.

In order to create a new file system, the **mkfs** command must have the following information:

dev The name of the device on which the file system is to be created.

size The size of the new file system in 512-byte blocks.

vol A volume label.

cyl The number of 512-byte blocks per cylinder on the disk. This parameter is used with **skip** to form the **interleave specification**. For an explanation of the interleave specification, see **mkfs** in *AIX Operating System Commands Reference*.

skip Used with **cyl** to form the **interleave specification**.

boot The name of a file containing a **bootstrap** (initialization) program to be loaded into the boot block of the new file system.

Note: "Information about File Systems—The /etc/filesystems File" on page 2-36 discusses these and other file system attributes more fully.

You can supply these parameters to **mkfs** in one of three ways:

- From information contained in **/etc/filesystems**.
  1. Use a text editor to create a stanza for the new file system in **/etc/filesystems**. The most convenient way to create this stanza is to copy a similar stanza, give it the label you want to assign to the new file system, and then modify it as necessary. Save the modified **/etc/filesystems** file.

2. At the $ (shell) prompt, enter the command `/etc/mkfs` *label*.

A stanza in **/etc/filesystems** contains, in one place, all of the information that defines that file system. Thus, if you need to know something about how a file system is defined, it is very easy to get the information you need if the file system has a stanza in **/etc/filesystems**.

- On the command line along with the **mkfs** command. For information on supplying parameters in this way, see **mkfs** in *AIX Operating System Commands Reference.*
- From the special file for the device on which the file system is to be created. If you do not supply the parameters for the new file system in one of the ways just described, **mkfs** takes them, if possible, from the device driver for the device on which the file system is to be created. If you want to take the parameters from a device driver, simply specify the path name for the device on which you want to create the file system. For example, the following command creates a file system on the **/dev/hd7** device:

```
/etc/mkfs /dev/hd7
```

You cannot create a file system on a device that is not already defined in **/dev**.

Warning: Creating on a device not defined in **/dev** will erase all the data on the device.

Mounting and Unmounting File Systems

File systems are independent from each other and from the operating system. Each file system is associated with a different **device** (a minidisk or a diskette). Before you can use a file system, it must be connected to the existing directory structure (either to the root file system or to another file system that is already connected). The **mount** command makes this connection. During the start up process, the system automatically mounts the file systems with the line `mount = true` in their **/etc/filesystems** stanza. However, you must use the **mount** command to mount any file system that:

- Does not have a stanza in **/etc/filesystems**
- Has the line `mount = false` in its **/etc/filesystems** stanza.

The **mount** command attaches one file system to another at a specified directory.

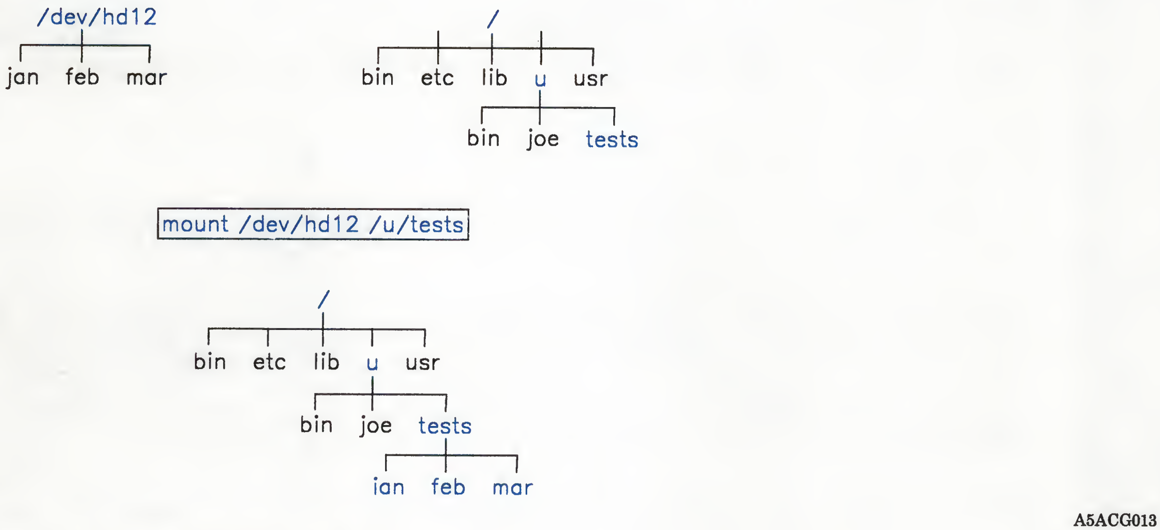


Figure 2-3. Mounting a File System

The general format for the **mount** command is:

`mount` *directory*

where *directory* is the name of the directory on which the file system is to be mounted. For example, to mount a file system identified in **/etc/filesystems** as `/controls` (`mount = false`), enter:

```
mount /controls
```

If you have superuser authority, you also can specify the device that contains the file system to be mounted. The format for **mount** when you specify both the device and directory is:

`mount` *device directory*

For example, to mount the file system on device **hd9** on the directory `controls`, use the command:

```
mount /dev/hd9 /controls
```

If there is a stanza in **/etc/filesystems** for the directory to be mounted, simply specify the name of the stanza with the **mount** command. For example, **mount /u** mounts the file system described by the following **/etc/filesystems** stanza onto the **/u** directory:

```
/u:
        dev       =  /dev/hd1
        vol       =  "/u"
        mount     =  true
        check     =  true
        free      =  true
```

Notice that the **mount** attribute in this stanza is set to **true**, which means that this file system is automatically mounted by the **mount all** command in **/etc/rc**. Thus, under normal circumstances, you do not have to use **mount** to mount this file system.

After you mount a file system, *directory* functions as the root directory of that file system. Also, the operating system records the presence of this file system in the file **/etc/mnttab**.

Use the **mount** command by itself (without *device* or *directory*) to list what file systems are mounted, where they are mounted, when they were mounted, and whether they are **writable** (that is, whether their contents can be modified).

You can disconnect mounted file systems with the **umount** command. The general format for the **umount** command is `umount` *filename*, for example:

```
umount /dev/fd2/accounts
```

where *filename* is either the special file for a mounted device or the name of the directory on which a device is mounted. To unmount all mounted file systems, use the command `umount all`.

Creating and Mounting Diskette File Systems

In general, the procedures for creating and using file systems are the same, whether the file systems reside on minidisks or diskettes. However, before you use diskette file systems, you should be aware of the information in this section.

Notes:

1. If you are not familiar with the proper way to handle diskettes, please read the material on diskette handling in *Guide to Operations* before you begin to use diskettes for AIX file systems.
2. If you want to transfer data to or from a DOS-formatted diskette, see "Using DOS Formatted Diskettes" on page 2-44.

Formatting Diskettes

Before you can create a file system on a diskette, you must use the **format** command to prepare the diskette to contain data.

Note: Formatting erases any data stored on the diskette. Following is the procedure for formatting diskettes:

1. Insert a diskette into the diskette drive. If you have more than one diskette drive, insert the diskette into drive 0 (A).

2. Enter: `format`

For more information about the **format** command, see **format** in *AIX Operating System Commands Reference.*

Creating Diskette File Systems

After you have formatted a diskette, you must create a file system on it:

1. Insert a formatted diskette into the diskette drive (or simply leave the diskette in the drive if you have just formatted it). If you have more than one diskette drive, insert the diskette into the drive 0 (A).

2. Enter: `/etc/mkfs /dev/fd0`

Mounting and Unmounting Diskette File Systems

To use a diskette file system, insert the diskette into the diskette drive and enter a command of the form:

`mount` *directory*

The standard directory for mounting a diskette file system is **/diskette0**. (If you have a second diskette drive, it is associated with the directory **/diskette1**.)

Note: Use an editor to delete the comment symbols (*) from the `diskette/:` stanza in **/etc/filesystems** before you mount it this way the first time.

Warning: Do not remove a mounted diskette. Damage to the diskette file system could result.

When you finish your work with the diskette file system, unmount it with a command of the form:

`umount` *directory*

In the **/etc/filesystems** stanza for **/diskette0**, the value for **mount** is **false**. Thus, the system does not automatically mount the diskette file system at startup.

If you want to mount a diskette on a directory other than **/diskette0**, use a **mount** command of the form: **mount** *device directory*. For example, if you want to mount a diskette file system on the directory **/mnt**, insert the diskette in drive 0 and enter:

```
mount /dev/fd0 /mnt
```

Using DOS Formatted Diskettes

With the **dosread** and **doswrite** commands, you can use diskettes to transfer data between the AIX system and a computer that uses the Disk Operating System (DOS). The **dosread** command copies a DOS file from the diskette to the AIX file system. The **doswrite** command copies a AIX file to a DOS diskette. Both commands perform the necessary translations required to make the data usable on the destination system.

The most common form of the **dosread** command is:

`dosread -a` *dosfilename filename*

where:

-a specifies translations usually required for text files.

dosfilename is the name of the file you want to copy from the DOS diskette.

filename is the name you want to copy the file to in the AIX file system.

The most common form of the **doswrite** command is:

`doswrite -a` *filename dosfilename*

where:

-a specifies translations usually required for text files.

filename is the name of the file you want to copy from the AIX file system.

dosfilename is the name you want to copy the file to on the DOS diskette.

For more information, see **dosread** and **doswrite** in *AIX Operating System Commands Reference.*

Backing up Files and File Systems

Once your AIX system is set up and in use, your next consideration should be backing up the file systems (there is one file system per minidisk). Whether due to system malfunction or user error, file systems and the data they contain can be damaged or lost. If you take a careful and methodical approach to backing up your file systems, you should always be able to restore recent versions of files or file systems with little difficulty. This section discusses different file and file system backup procedures and how those procedures can be combined into a dependable backup policy.

Types of Backups

Backup procedures rely upon the following commands:

cvid Use the **cvid** command to backup the VRM minidisk. You should always have a backup of the current VRM minidisk. Before you modify the VRM minidisk (for example, by installing a licensed program or an update, or using the **mvmd** command), make certain that you have a backup of the current VRM minidisk. If not, backup the VRM minidisk before you make the changes. Always make a new backup of the VRM minidisk after you modify the VRM minidisk. The VRM minidisk backup procedure is explained under "Backing Up the VRM Minidisk" on page 2-47.

backup Use the **backup** command to backup individual files or entire file systems. "Individual File Backup and Restore" on page 2-52 explains how to backup individual files. "Using the backup and restore Commands" on page 2-48 explains how to backup complete file systems with the **backup** command. You can also backup from the AIX Operating System Installation/Maintenance diskette. The **USE MAINT CMDS** option allows you to back up by inode, and the **USE STANDALONE SHELL** option allows you to run the **backup** command for a back up by name. For information about the maintenance system, see "Running the Maintenance System" on page 2-5.

restore Use the **restore** command to read files backed up with **backup** to a device or directory. For information on how to restore individual files, see *Using the AIX Operating System*. "Using the backup and restore Commands" on page 2-48 explains how to restore complete file systems. (The **restore** command can restore individual files from a complete file system backup.)

To use **restore** from the maintenance system, select `Restore commands` from the **USE MAINTENANCE COMMANDS** menu, and then select `Restore a file system`.

Note: When you select `Restore a file system` from the **USE MAINTENANCE COMMANDS** menu, the restore is by i-node only.

(For information about the maintenance system, see "Running the Maintenance System" on page 2-5.)

dd Use the **dd** (device-to-device copy) command to backup entire file systems. The **dd** command is a faster way to backup entire file systems. However, you cannot restore individual files from a **dd** backup. "Backing Up Complete File Systems with the dd Command" on page 2-55 explains how to perform file system backups with the **dd** command.

You also can use **dd** from the **USE MAINTENANCE COMMANDS** menu to:

- Backup a file system.

  Select `Backup commands` and then select `Back up a minidisk image`.

- Restore a file system.

  Select `Restore commands` and then select `Restore a minidisk image`.

(For information about the maintenance system, see "Running the Maintenance System" on page 2-5.)

It is very important for you to understand how to use these commands to protect the users of your system from loss of data. You should be familiar with the information in:

- "Running the Maintenance System" on page 2-5
- "Backup Media" on page 2-47
- "Backing Up the VRM Minidisk" on page 2-47
- **cvid** in *AIX Operating System Commands Reference*
- "Using the backup and restore Commands" on page 2-48
- **backup** in *AIX Operating System Commands Reference*
- **restore** in *AIX Operating System Commands Reference*
- "Backing Up Complete File Systems with the dd Command" on page 2-55
- **dd** in *AIX Operating System Commands Reference.*

Backup Media

Diskettes are the standard backup medium. Unless you specify a different value for **backupdev** in **/etc/filesystems**, the **backup** command automatically writes its output to **/dev/rfd0**, the device name for the diskette drive.

When you install a streaming tape drive, the **devices** command automatically changes the value of **backupdev** to **/dev/rmt0**, the device name for the tape. The **rmt0** value makes the tape rewind when it is finished backing up a file system. If you change the value to **rmt4**, the tape does not rewind.

Note: You may need to change the value for **backuplen**. For example, a **backuplen** of 2700 represents a 300-foot tape with nine tracks (300x9=2700).

If you remove a tape drive from the system, **devices** changes the value of **backupdev** back to **/dev/rfd0** and the value of **backuplen** back to 2400.

After you install a tape device, the **backup** command uses the tape as the backup device for complete file system backups (the procedure described under "Using the backup and restore Commands" on page 2-48). However, even with a tape installed, the **backup** command still backs up individual files to diskettes (for information about file backups, see *Using the AIX Operating System*).

Backing Up the VRM Minidisk

Use the **cvid** command to backup your VRM minidisk files onto a diskette. You should always have a backup of your current VRM. With a current VRM backup, you are prepared to reinstall the most recent version of the VRM should a system failure or other error require it.

Backup the VRM immediately after you install it and back it up again after each change you make to it. The following commands can modify the VRM, and you should backup the VRM after using any of them:

installp Modifies the VRM when you install a licensed program.

updatep Modifies the VRM when you install an update.

mvmd Modifies the VRM directly, by adding, changing and deleting files. Generally, **mvmd** is used by **installp** and **updatep**; that is, you do not ordinarily enter **mvmd** commands on the command line.

(For more information on **installp**, **updatep**, and **mvmd**, see *Installing and Customizing the AIX Operating System* and *AIX Operating System Commands Reference*.)

This section describes how to use **cvid** to create a VRM diskette. You must have superuser authority or be a member of the system group to use the **cvid** command. (See *AIX Operating System Commands Reference* for a discussion of the **cvid** command. For an

explanation of any error messages that may occur as you use this command, see *Messages Reference.*)

Note: Before you change the VRM, make certain that you have a backup of the current VRM. If not, backup the VRM before you change it.

The command format is:

```
cvid dev -f -v proto
```

where:

dev Specifies the device on which the file system is created, as in fd0 (diskette drive 0). It can also be the name of a file system.

-f Specifies the file system label for the new file system. The default label is **/vrmmnt**.

-v Specifies volume label for the new file system. The default volume label is **IBMVRM**.

proto Specifies the name of the prototype file. Default is **/vrm/vproto**.

To Back Up Your VRM Minidisk

1. Insert a formatted diskette into the diskette drive.
2. After the system prompt, type `cvid`, plus any needed parameters.
3. Press **Enter**.
4. Insert the second diskette when prompted.
5. When the VRM files have been copied to the diskettes, remove the new VRM install diskettes and store them in a safe place.

While the **cvid** command is running, the system creates a file system, using the **mkfs** command, and copies files from your VRM minidisk.

Using the backup and restore Commands

The **backup** command allows you to back up files selectively or to backup entire file systems. Similarly, you can use the **restore** command to restore all or part of the backups you create with **backup**.

Note: To back up by filesystem or minidisk, you must have read permission to both the block device **/dev/hdx** and the character device **/dev/rhdx**. To restore a filesystem, you must have write permission to both the block and character devices.

Both **backup** and **restore** are available from the **USE MAINTENANCE COMMANDS** menu in the maintenance system. For information on the maintenance system, see "Running the Maintenance System" on page 2-5.

This section explains how to use **backup** and **restore** for complete file systems. "Individual File Backup and Restore" on page 2-52 explains how to backup and restore specific files within a file system.

Note: If you use diskettes as your backup media, have several formatted diskettes ready to use before you enter the **backup** command.

Note: The time and date information recorded for a backed up file reflect when the backup was made. Ordinarily, the **restore** command preserves this time and date information. However, if you want restored files to reflect when they were restored (rather than when they were backed up), use the **-h** flag with **restore**.

To Back Up a File System with backup

1. Prepare the backup medium:
   - Make certain that the tape device is ready to operate, or
   - Insert a formatted diskette into drive 0.
2. Enter:

   `backup -0 -u` *filesystem*

where *filesystem* is the name of the file system to be backed up.

Note: If you back up to tape you should use the **-f** flag with **backup**.

(The system prompts you for additional backup media as required).

To Restore a File System

1. Load the tape or diskette containing the file system to be restored. (If you restore from diskettes, and the backup occupies more than one diskette, insert the first diskette of the group into drive 0).
2. Enter:

   `restore -r` *filesystem*

   Note: If you restore to tape you should use the **-f** flag with the **restore** command.

   where *filesystem* is either the name of a physical device or a directory name listed in the **/etc/filesystems** file. The **-r** flag is for i-node backups only.

(The system prompts you for additional media as necessary, for example, for the second of a group of diskettes.)

Volume Backups

The **backup** command individually backs up each file in a file system. Thus, you do not have to restore an entire file system if all you need to do is restore specific files. Also, you can restore a file or group of files to any file system that is large enough to accommodate them.

To backup an entire file system, use a command of the following form:

`backup -0 -u` *filesystem*

Following is an explanation of the parts of this command:

-0 A flag that indicates the **level** of the backup. Level **0** causes every file in the file system to be backed up. Use other backup levels (1-9) for incremental dumps, as is explained under "Incremental Backups" on page 2-51.

-u A flag that causes **backup** to record the date and level of the backup in the **/etc/budate** file. **/etc/budate** contains the date of the last backup of each file system at each level.

filesystem The name of the file system to be backed up. *filesystem* can be either the name of the physical device that contains the file system or the name of the file system's root directory.

Unless you specify otherwise on the command line, the **backup** command uses information in **/etc/filesystems** to determine:

dev The device, or minidisk, that contains the file system.

backupdev The device (for example, a diskette drive) on which the backup is to be made.

backuplen The size in blocks of the backup device (diskette or tape) to be used (for computing the amount of data that will fit on each diskette).

Notes:

1. When you restore a complete file system, the **restore** command organizes the files efficiently, but does not reduce the total amount of storage space needed for the file system. The **-m** flag backs up an entire minidisk as an exact image. An image backup is appropriate for backing up very large AIX file systems or minidisks that do not contain AIX file systems. A backup by minidisk (**backup -m**) must be restored by minidisk (**restore -m**).

2. It is recommended that you use exact minidisk sizes when restoring with the **restore -m** command. When restoring a minidisk, AIX prevents you from restoring to a disk that is too small. AIX does not prevent you from restoring to a disk that is larger than required. When this happens, the extra space on the minidisk becomes inaccessible.

Incremental Backups

A level 0 backup can require a considerable amount of time and a large number of diskettes to complete. Because only a fraction of the files on your system can change from day to day, it is not necessary for you to make a level 0 backup daily. Instead, you can make **incremental** backups—backups of only the files changed since a previous backup.

An incremental backup backs up all files changed since the last backup at the next lower level. For example, the following level 1 **backup** command backs up all files changed since the last level 0 backup:

`backup -1 -u` *filesystem*

The parameter that indicates the level of the backup (`-1` in this example) can be any number from 0 through 9. Thus, a level 4 backup backs up all files that have changed since the last level 3 backup, and so forth. You can use the backup levels to develop a very dependable backup system. For example, you might make level 0 backups monthly, level 1 backups weekly, and level 2 backups daily. Then, should you have to restore a complete file system, you would restore from the latest level 0 backup, then from the latest level 1 backup, and finally from the latest level 2 backup. For many systems, it should be adequate to use only two levels of backup (for example, level 0 weekly and level 1 daily or level 0 monthly and level 1 weekly).

Individual File Backup and Restore

You can also use the **backup** command to backup individual files, for example, when some or all of the files that belong to a particular user will not be needed for an extended period. Backup copies of individual files are also a convenient means for exchanging data among different systems (for example, you backup the files from your system onto a diskette and then the user of another system restores the files from the diskette). To backup individual files, use the **-i** (backup by name) flag with the **backup** command. To restore files backed up by name, use the **-x** (restore named files) with the **restore** command.

The following sequence backs up three files to a backup device defined in **/etc/filesystems**:

```
$ backup -i
Please mount volume 1 on /dev/backup device
. . . and press enter to continue
file1
file2
file3
END OF FILE
Done at date and time
n blocks on n volume(s)
$ _
```

You can also use the **backup** command with a list of file names created with an editor. In the following example, **backup** makes backup copies of the files named in the file `list`:

```
backup -i < list
```

To backup all files and subdirectories of the current directory, use the command:

```
find . -print | backup -i
```

To restore files that have been backed up by name, use the command:

```
restore -x
```

If your backup device is a streaming tape drive, you can perform faster **backup -i** and **restore -x** operations by using these commands in a pipeline with the **dd** command. For example, the following pipeline pipes the output of **find** to **backup**, and then pipes the output of **backup** to **dd**:

```
find . -print | backup -if- -C30 | dd of=/dev/rmt0 bs=30b
```

In this example, the **f-** flag causes **backup** to write to standard output, and **-C30** specifies the number of blocks in a single output operation (the cluster size). The **of=/dev/rmt0** parameter causes **dd** to write to the streaming tape device, and **bs=30b** specifies the size of a single input or output operation (30 blocks). You may determine that other values for **-C** and **bs=** give better performance in your application.

To restore files backed up by piping the output of **backup** to **dd**, use a pipeline of the form:

```
dd if=/dev/rmt0 bs=30b | restore -xf-
```

In this example, **if=/dev/rmt0 bs=30b** causes **dd** to read from the tape device with an input and output operation size of 30 blocks. The flags to the **restore** command cause it to restore by name (**-i**), reading from standard input (**f-**).

Guidelines for Backup Policies

No single backup policy can meet the needs of all AIX system users. A policy that works well for a system with one user, for example, could be inadequate for a system that serves 5 or 10 different users. A policy developed for a system on which many files are changed daily would be inefficient for a system on which data changes infrequently. Only you can determine the best backup policy for your system, but the following general guidelines should help:

- **Make sure you can recover from major losses.**

  Can your system continue to run after any single fixed disk fails? Can you recover your system if all of the fixed disks fail? Could you recover your system if you lost your backup diskettes or tape to fire or theft? Although these things are not likely, any of them is possible. Think through each of these possible losses and design a backup policy that would enable you to recover your system after any of them.

- **Use your backups periodically**.

  Diskettes, diskette drives, and tape devices can be unreliable. A large library of backup tapes or diskettes is useless if their data cannot be read back onto a fixed disk. Thus, it is a good idea to make certain that your backups are usable. Try to restore files periodically (for example, to **/dev/null**), just to make sure that they can be read. If you use diskettes for your backups and have more than one diskette drive, try to read diskettes from a different drive than the one on which they were created. Therefore, you may want the security of repeating each level 0 backup with a second set of diskettes. If you use a streaming tape device for backups, you can use the **tapechk** program to perform rudimentary consistency checks on the tape. For more information about the **tapechk** command, see **tapechk** in *AIX Operating System Commands Reference*.

- **Keep old backups**.

  You probably will develop some cycle for re-using your backup media—tapes or diskettes. However, you should not re-use all of your backup media. Sometimes it may be months before you, or some other user of your system, notices that an important file is damaged or missing. You should save old backups for just such possibilities. For example, you could have three cycles of backup tapes or diskettes:

  - Once per week, recycle all daily diskettes except the one for Friday.
  - Once per month, recycle all Friday diskettes except for the one from the last Friday of the month. This makes the last four Friday backups always available.
  - Once per quarter, recycle all monthly diskettes except for the last one. Keep the last monthly diskette from each quarter indefinitely, perhaps in a different building.

- **Check file systems before backing them up.**

  A backup that was made from a damaged file system may be useless. Before making your backups, it is good policy to check the integrity of the file system with the **fsck** command (covered under "Maintaining the File System" on page 3-4).

Your system should not be in use when you make your backups. If you backup a file system while it is in use, files can change while they are being backed up. The backup copy of such a file would not be accurate.

Finally, it is always good policy to backup your entire system before any hardware testing or repair work is performed or before you install any new devices, programs, or other system features.

Backing Up Complete File Systems with the dd Command

When you want to backup (and restore) only complete file systems, the **dd** (device-to-device copy) command gives you a potentially faster alternative to **backup** and **restore**. The **backup -m** command actually uses **dd** and is a convenient way to backup file systems. However, this section describes how to use the **dd** command.

The **dd** command is available from the **USE MAINTENANCE COMMANDS** menu in the maintenance system. To use **dd** from the maintenance system:

- To backup a file system, select `Backup commands` and then select `Back up a minidisk image`.
- To restore a file system, select `Restore commands` and then select `Restore a minidisk image`.

For information on the maintenance system, see "Running the Maintenance System" on page 2-5.

To backup a file system with **dd**, you need to specify the following parameters:

if The name of the input file. Use the value of **dev** = in the **/etc/filesystems** stanza for the appropriate file system. **dd** works more quickly with the raw versions of devices (that is, the ones which begin with **r**).

of The name of the output file. Use the device name for the streaming tape device (**/dev/rmt0**).

bs The input and output block sizes, in bytes.

The following **dd** command backs up 80 blocks from the file system on `/dev/hd`*n* to the streaming tape, `/dev/rmt0`, using a block size of **512** bytes:

```
dd if=/dev/rhdn of=/dev/rmt0 bs=512 count=80
```

To restore a file system backed up with **dd**, use the **dd** command again, copying the file system from the backup device to the minidisk.

Note: The **dd** command may be less efficient than the **backup** command if the file system being backed up does not contain many files. Also, if you restore a backup made with **dd** to a different minidisk, its size may change.

Understanding System Security

The key to effective security is understanding how the security features work and then using them conscientiously. If more than one person uses your system, it is important for everyone to understand how security works and the importance of observing certain security guidelines.

The principal AIX security features are described in other sections of this book. This section summarizes important security features and indicates where you can find more detailed information.

Passwords

Passwords help protect your system against unauthorized access. The AIX system encrypts passwords, stores them in a file, and then compares the password supplied when a user tries to log in with the encrypted version. If the two match, the user gains access to the system.

While it is not mandatory to use passwords on your system, it is strongly advised, even if you are the system's only user. Without password protection, all data on your system is available to anyone who knows how to turn on the power switch and enter a valid user name (generally, user names are easy to obtain or guess). Even if your system does not contain sensitive data, an unauthorized user cause problems by altering the contents of files or turning off the system without running **shutdown**.

Note: If user **root** does not have a password, password protection is not enforced. Anyone who knows a valid user name can log in without entering a password and use the AIX system.

For more information about passwords, the password file, and the **passwd** command, see the following:

- *Using the AIX Operating System*
- "Managing User Accounts" on page 2-13
- **passwd** in *AIX Operating System Commands Reference*
- "Configuring Password Restrictions" on page 6-12.

File Protections

Every file and directory in the AIX system has a group of permissions associated with it. The permissions define who can use the file in what way. In other words, the permissions grant certain users access to the files and directories and protect the files and directories from other users.

For an explanation of how the file and directory permissions work, see *Using the AIX Operating System*. While permissions afford an effective way to control access to data stored in the system, they are only as effective as users make them. Thus, it is important for all users of your system to understand how the permissions work, how they can be modified, and how their effectiveness can be undermined by a lax attitude toward system security.

Invalid Login Attempts

Invalid login attempts due to an incorrect login name or password are recorded in the **/etc/.ilog** file. The contents of this file can be examined only by the user **root** or **su**, or a member of the system group. Each time you log in as **root** or **su** and there is an entry in **/etc/.ilog**, the system displays a message advising you to check the contents of **/etc/.ilog**. Use the **who** command to look at this file. See “Other System Log Files” on page 6-30 for additional information about this file.

Terminal Logging

The terminal logging daemon, **tlogger**, collects all data read or written to the associated terminal, except passwords, and writes the data to a log file. If standard error is a terminal device, it is used as the associated terminal; otherwise the process's **usrinfo** is used to identify the login tty and that device is used as the associated terminal. You can log data to a terminal other than the login terminal by redirecting standard error. You can also have multiple **tlogger** commands running at the same time if each logs data for a different tty device, and each uses a different log file.

To start **tlogger** enter:

```
/etc/tlogger &
```

Flags available with **tlogger** are:

-c*file* Specifies *file* as the name of the log file. The log file defaults to **/usr/adm/ras/tlogfile**.

-b*file* Specifies *file* as the name of the back-up log file. The back-up log file defaults to **/usr/adm/ras/tlogfile.bk**.

Each time that you start **tlogger** the current log file is written to the back-up file and a new log file starts with the permissions set to allow read and write by the owner.

Two commands, **tlog off** and **tlog on**, let you stop and then restart sending data to the log file. Enter `tlog off` to stop sending data to the log file while you are logged on the terminal. The **tlogger** daemon continues to run but your activities not logged to the log file. Before you log off of the terminal enter `tlog on` to restart the logging process. The **tlogger** daemon continues to run even though you are not logged on; this provides information on any console activity while you are not logged on.

You can end **tlogger** with the **kill** command or it ends when shutdown occurs. **tlogger** also ends if the log file becomes full and no data can be written to it. You should not use SIGKILL to stop **tlogger** since system resource clean-up cannot be performed.

Chapter 3. Maintaining the AIX Operating System

CONTENTS

About This Chapter

This chapter covers the tasks required to maintain the AIX Operating System and to adapt the system to your needs. Anyone who manages an AIX system will perform some of the tasks described in this chapter (for example, maintaining the file system). Other tasks described here only concern you if you modify your system (for example, installing applications and local commands). A third set of tasks may become more important the longer you use the system (for example, refining your security policies).

Maintaining the File System

Each time a file is created, changed, or deleted, the operating system performs a series of file system updates. These updates, when written to a disk, produce a ***consistent*** file system. If all of the updates do not complete successfully, the result can be an inconsistent file system. Some inconsistencies may not be severe, but others, if not promptly corrected, can spread and eventually make an entire file system unusable. The **fsck** (file system consistency check) command analyzes a file system, locates inconsistencies, and often can correct certain inconsistencies automatically.

Causes of File System Damage

The major causes of file system damage are:

- A system halting (or failing) before it completes all pending file system updates, for example, due to power failure
- Poor operating procedures
- Failure of a disk drive or drive controller
- Physical damage to a diskette.

Disk Buffering—the sync Command

Any operation involving a file requires data to be retrieved from the fixed disk (read into memory), returned (written) to the disk, or both. These read and write operations also are known as disk input and output, or ***disk I/O***. To make some disk I/O more efficient, the operating system maintains a ***buffer*** (temporary storage area) in memory for recently accessed data blocks. The operating system writes data from the buffer to the disk only when:

- There is a **sync** system call.
- There is an **fsync** system call.
- The system needs the buffer for another purpose.

Disk buffering can increase the efficiency of I/O operations because it allows repeated reads and writes to the same block to occur without the block being physically read from the disk or written to the disk each time. The system is also able to schedule certain I/O operations (especially read operations) so that they occur in an order that is more efficient for the disk, rather than in the order in which they were requested. Together, disk buffering and disk scheduling help match the demand for I/O operations to the physical characteristics of the fixed-disk.

Although it may improve the efficiency of disk I/O, disk buffering can indirectly cause one type of file system damage—if the system stops before writing all data from the buffer back

to the disk, the data on the disk is not current (and the data in memory is lost). Thus, it is extremely important for buffered data to be written to the disk (with the **sync** command) before the system is stopped. The **shutdown** command automatically runs the **sync** command. Failure to run **sync** (with **shutdown**) before stopping the system is a common cause of file system damage, but one that is very easy to avoid.

Even if the system fails, which usually means that you do not have time to run the **sync** command, any structural damage to the file system is usually easy to find and repair. The **fsck** command often can automatically repair this kind of damage. Thus, before you use the file system, you check it with **fsck** (either use the **Check a file system** item on the AIX Operating System Installation/Maintenance diskette, or run **fsck** from the standalone shell, as described under "Running the Maintenance System" on page 2-5). Otherwise, if files are created on the damaged file system, the damage can become much worse.

The **dfsck** command lets you check two file systems on two different drives at the same time. Flags specified with **dfsck** are passed to **fsck**, letting you interact with two **fsck** programs. **dfsck** prints the file system name for each message that it sends. Thus, when you answer a message from **dfsck**, prefix the response with either 1 or 2 to indicate the first or second file system. For more information about **dfsck**, see "**fsck**" in *AIX Operating System Commands Reference.*

Note: The **fsck -p** command should be included in **/etc/rc** so that it always runs during normal startup. To check two file systems on two different drives at the same time during normal startup, include the **dfsck** command in **/etc/rc**.

Examples

Following are two examples of how a file system can become inconsistent. In each example, the system stops before all disk updates are complete.

- Example 1. Block Allocation Inconsistency

  When the operating system allocates a new block to a file, two things must be updated:

  - The i-node, to indicate that the file contains a new block
  - The free list, to indicate that a block that was free is no longer available.

  If the system stops between the first and second update operations, the i-node shows that the new block is allocated to the file, while the free list still shows that the block is available. There is an inconsistency in the file system.

  The **fsck** command often can find and correct such inconsistencies if it is run on the file system immediately. However, if you begin to use the file system again without correcting the inconsistency, more serious problems can occur. For example, the block shown as both allocated (in the i-node) and free (in the free list) could be allocated to a second i-node; that is, the same block would be part of two different files or directories. File system damage of this type can spread quickly and be extremely difficult to correct; it can result in the appearance of strange information in files and the loss of some files or even an entire file system.

- Example 2. File Deletion Inconsistency

  When the operating system deletes a file, up to five things must be updated:

  – The directory entry that points to the file must be cleared.

  – The reference count in the i-node for the file must be reduced by one.

  – If the reference count goes to zero, all blocks in the file must be added to the free list.

  – If a large number of blocks are freed, a new link in the free chain must be written.

  – The i-node for the file must be made available again.

  Depending upon which of these update operations is completed at the time the system stops, different types of file system inconsistencies can occur.

Note: Lost data does not necessarily produce file system inconsistency. Generally, the data stored in ordinary files has nothing to do with the consistency of the file system. For example, if you begin a **write** operation and the system stops before the operation completes, the file on the disk simply does not get updated. While you have lost data, that loss may not affect the file system. The **fsck** command can only determine whether the structure of the file system is internally consistent; it cannot check the data within files.

Checking and Repairing File Systems—The fsck Command

A file system contains redundant information about its structure. If the file system is consistent, each instance of a particular piece of information is identical. The **fsck** command uses this redundant information to detect and, in many cases, correct file system inconsistencies.

The **fsck** program makes several passes through a file system, performing a different phase of checking in each pass. When it detects an inconsistency, **fsck** reports the error condition, displaying a message on the screen and waiting for a response. For an explanation of the **fsck** messages and what actions you should take to respond to them, see *Messages Reference.*

The file **/etc/rc** contains a list of the commands run each time the system starts. Generally, **/etc/rc** contains the **/etc/fsck -p** command. This line starts **fsck** before the system mounts any file systems. If the **/etc/filesystems** stanza for a file system contains either of the following lines, **fsck** checks the file system at this point:

```
check=true
```

check=*group*

(**fsck** checks all file systems with the same *group* at the same time).

Note: A ***stanza*** is a group of lines that define a part of the system. A stanza in **/etc/filesystems** defines a file system.

The **-p** (preen) flag causes **fsck** to run automatically, correcting all simple inconsistencies without requiring any action from the user. The **-p** flag enables **fsck** to repair most inconsistencies that result from the system stopping unexpectedly.

Without the **-p** flag, **fsck** runs interactively, displaying each error condition that it locates and waiting for you to decide whether corrections should be made. For information about other **fsck** flags, see "**fsck**" in *IBM RT AIX Operating System Commands Reference.*

It is very important to run **fsck** on unmounted file systems. The **fsck** command goes completely through the file system several times, often comparing data collected on one pass with data collected on another. Thus, any activity on the file system can interfere with the ability of **fsck** to check and repair file system damage.

The fsck Consistency Checks

Following is a list of the types of file system inconsistencies that **fsck** checks for, together with descriptions of how they can be corrected:

- Superblock Inconsistencies

  The superblock is the most frequently updated part of a file system and it is therefore the most likely to be inconsistent after the system stops unexpectedly. Every time a block or i-node is allocated or deallocated, the superblock should be updated.

  The **fsck** program checks all information about the file system in the superblock to make sure that it is consistent with the standard requirements for a file system. This information (including file system size, number of blocks allocated to i-nodes, cluster size, and interleave factors) never changes during normal operation. Should any of these values change, there is evidence that the file system is severely damaged. In such cases, **fsck** reports the condition and does nothing else with the file system. Usually the only way to recover such a file system is to restore it from a backup.

  If the information about the file system seems to be consistent and reasonable, **fsck** uses the file system and i-list size to validate all block numbers. All reasonable blocks are between the end of the i-list and the end of the file system. Any reference to a block not within that range is not valid and probably indicates severe file system damage.

- Free block list

  The free list is a series of logically connected blocks which are available (that is, they are not allocated to any file). The superblock contains the first block of the free list. The **fsck** command checks the structure of the free list and also checks the block numbers of every block in the list. **fsck** makes separate checks to make sure that no allocated blocks appear in the free list and that all blocks that are not allocated do appear in it. **fsck** can repair any inconsistencies found in the free list, usually by constructing a new free list that contains all blocks found not to be allocated.

- Free block count

  The superblock contains a count of the total number of free blocks within the file system. **fsck** compares this count to the number of blocks it found free within the file system. If the counts disagree, **fsck** can replace the count in the superblock with the actual free block count.

- Free i-node count

  The superblock contains a count of the total number of free i-nodes within the file system. The **fsck** command compares this count to the number of i-nodes it found free within the file system. If these counts disagree, **fsck** can replace the count in the superblock with the actual free i-node count.

• I-node Inconsistencies

  An individual i-node is not likely to be inconsistent as is the superblock. There is, however, a danger that the i-nodes for files that are in use when the system stops will be inconsistent.

  The **fsck** command checks the list of i-nodes in sequence, starting with i-node 1 and continuing to the last i-node in the file system. **fsck** checks each i-node for inconsistencies in format and type, link count, duplicate blocks, bad blocks, and i-node size.

  - I-node format and type inconsistencies

    Each i-node contains a ***mode word***—a description of the type and state of the i-node. There are five i-node types:

    Regular
    Directory
    Block special
    Character special
    First-in-first-out (FIFO, also called a named pipe).

    An i-node whose type is not one of these five does not have a valid type.

There are three i-node states:

Allocated
Unallocated
Neither allocated nor unallocated.

An i-node that is neither allocated nor unallocated is not correctly formatted. The format of an i-node can be damaged when incorrect data are written to the i-node list (for example, as a result of a system failure).

– I-node link count inconsistencies

Each i-node contains a count of the total number of directory entries that refer to the i-node (the ***reference count***). **fsck** verifies the reference count in each i-node by scanning the entire directory structure, beginning with the root directory, and counting the actual number of links to that i-node.

If the stored link count is not zero and the actual link count is zero, the file is inaccessible. If the inaccessible file contains any data, **fsck** links the file into the **lost+found** directory on the file system. If the file contains no data, **fsck** deallocates the i-node for that file.

If the actual link count is neither zero nor equal to the stored link count, it is likely that a directory entry was added or removed without the i-node being updated. **fsck** can replace the stored link count with the correct value.

– Inconsistencies from duplicate and invalid blocks

Each i-node contains a list, or pointers to lists (indirect blocks), of all the blocks claimed by that i-node. **fsck** first determines whether each block number in the list is valid and then compares the block number against a list of allocated blocks. If it finds inconsistencies, **fsck** displays the names of all files that contain invalid or duplicate blocks.

Inconsistencies of this type usually occur when you create or extend files on a file system that has duplicate blocks in its free list. If the duplicate block contains data in both files, the damage is not severe. If the file system uses the duplicate block for a directory or an indirect block, however, severe inconsistencies in the file system structure may occur. Although such inconsistencies are serious, you can prevent them by routinely checking every file system with **fsck** before you mount it. (For example, do not use the **fsck -f** command in **/etc/rc**. The **-f** [fast] flag saves time by not checking file systems that were cleanly unmounted the last time they were used. The extra precaution of running **fsck -p** from **/etc/rc** reduces the chance that your system will mount a damaged file system.)

Generally, **fsck** corrects such inconsistencies by deleting the files with invalid or duplicate blocks, but not automatically—that is, you must respond to a prompt from **fsck** before it will destroy the files. Before allowing **fsck** to destroy such damaged files, you should examine the files to determine whether you can save any important data.

– Size inconsistencies

In each i-node is a ***size field*** that indicates the number of bytes in the file associated with the i-node. **fsck** can determine whether the number of bytes is consistent with the number of blocks allocated to the file. In the case of directories, **fsck** can also make sure that the size is a multiple of 16 (the length of a directory entry).

If it finds size inconsistencies, **fsck** displays a warning message. Size errors are not serious, but **fsck** does not have sufficient information to correct the size. You can correct the problem by copying the data from the old file into a new file and then deleting the old file.

- Indirect Block Inconsistencies

  There are three types of indirect blocks: ***single-indirect***, ***double-indirect***, and ***triple-indirect***. A single-indirect block contains a list of some of the block numbers associated with an i-node. Each entry in a single-indirect block is a data block number. A double-indirect block contains a list of single-indirect block numbers. A triple-indirect block contains a list of double-indirect block numbers.

  Indirect blocks are associated with a particular i-node. Thus, inconsistencies in indirect blocks directly affect the associated i-node. **fsck** can check for inconsistencies such as blocks that are already claimed by another i-node and block numbers that are outside the range of the file system. **fsck** finds and corrects indirect block inconsistencies just as it does inconsistencies between the i-node and data blocks.

- Data Block Inconsistencies

  There are two types of data blocks: plain and directory. Plain data blocks contain the information stored in a file. Because an ordinary file may contain any type of data, **fsck** cannot check the validity of the contents of a plain data block.

  Directory data blocks contain directory entries. Each directory data block can be checked for inconsistencies involving directory i-node numbers pointing to unallocated i-nodes, directory i-node numbers greater than the number of i-nodes in the file system, incorrect directory i-node numbers for the current and parent directories (. and .., respectively), and directories disconnected from the file system.

  If a directory entry i-node number points to an unallocated i-node, then **fsck** tries to remove that directory entry. This condition might occur if the data block containing the directory entry were modified and written to the disk, but the system stopped before the i-node could be updated.

  If a directory entry i-node number points beyond the end of the i-node list, **fsck** tries to remove that directory entry. This condition occurs if incorrect data are written into a directory data block.

  The directory i-node number entry for . should be the first entry in the directory data block. Its value should be equal to the i-node number for the directory itself.

The directory i-node number entry for .. should be the second entry in the directory data block. Its value should be equal to the i-node number for the parent directory (or the i-node number of the directory itself if the directory is the root directory).

The **fsck** command checks the internal connections of the file system in general. If it finds a file or directory not linked into the file system, **fsck** links the file or directory to the **lost+found** directory. This condition can be caused by i-nodes being written to disk and the system stopping before the corresponding directory data blocks are updated. Items linked to **lost+found** are identified by their i-node number (for example, if **fsck** finds a file whose i-node number is `1234`, it links the file to `lost+found/1234`).

Repairing File Systems by Destroying Files

Under some conditions, the only way **fsck** can restore consistency to a file system is to clear i-nodes—that is, by destroying files. For example, if a block is allocated to two files, **fsck** clears both files. This section explains how to keep your loss of possibly useful data to a minimum.

Before letting **fsck** destroy a file, be sure you know which file is to be destroyed and what is wrong with the file. Most files damaged when a system stops unexpectedly are temporary files and they usually are of little value. However, if it is necessary to destroy a permanent file, you should try to salvage its contents and, if the file belongs to another system user, notify that person that the file is to be destroyed. Before **fsck** destroys a file, it lists all the names linked to that file, and prompts you to determine whether each link to the file can be removed. Only when all links are removed can **fsck** destroy the file.

Often, the name of a file indicates whether the file's contents are important. For example, temporary files may have names like `ctm5a.P5024` and often are in special directories like **/tmp** or **/usr/tmp**. Like temporary files, editor backup files (files with a `.bak` extension) and object modules (files with a .o extension) usually are not important. However, it is likely that a file named `boss/employees/raises` is quite important, and you should make every effort to salvage its contents. If you cannot tell from its name whether a file is important, try to salvage its contents.

If you know the i-number of a file, you can find the name of the file with the **ncheck** program. For example, if **fsck** indicates that two files, i-numbers 408 and 1212, on the **/u** file system have a duplicate block, you could find the names of those files by entering:

```
ncheck -i 408 1212 /u
```

For more information about the **ncheck** command, see **ncheck** in *AIX Operating System Commands Reference.*

When a file contains one or two bad or duplicate blocks, you probably can salvage most of its contents by copying them into a new file before destroying the original (damaged) file. To copy the contents of the file, mount the file system temporarily (and **readonly**, if

possible) and use the **cp** command. It is safest to copy the file onto another (consistent) file system. If you want to copy the file within its original file system, first run **fsck** to make sure the free list is consistent.

You can destroy a damaged file in several different ways. If a file contains invalid or duplicate blocks, **fsck** removes all links to the file and then destroys the file by deallocating its i-node (assuming that you respond y to the **fsck** prompts). You can destroy a file directly with the **clri** (clear i-node) command. However, it is safer to let **fsck** destroy the file, since **fsck** automatically restores the file's data blocks to the free list; **clri** does not do this.

You should not use the **rm** (remove file) command to destroy files for two reasons:

- You must mount the damaged file system to run **rm**. Mounting the damaged file system can compound the damage.
- **rm** frees all bad and duplicate blocks. Thus, if you removed the damaged file with **rm**, the same type of damage is likely to occur the next time the system allocates those blocks.

Any time you destroy files, run **fsck** on the file system before using it again. **fsck** eliminates inconsistent directory references and makes sure that the free list is correct.

The Input/Output System

An I/O (input/output) system is a mechanism for transferring data among system devices (for example, when you print a file, the I/O system transfers the data from the file to the printer). The AIX I/O system is independent of the particular devices that produce output or receive input. If three devices (for example, a printer, a fixed-disk, and a display station) can all take input in the same form, the I/O system makes no distinction among them. To understand how the I/O system works, you should first have a general understanding of device drivers and special files.

Device Drivers and Special Files

A ***device driver*** is a program that makes the logical connection between the system and a device. The kernel contains a device driver for each type of device on the system. For example, if your system has two identical printers, you need only one device driver to run both of them; if you have two different types of printers, your system needs a separate device driver for each printer type.

Each device has a ***class***, a ***major device number***, and a ***minor device number***. There are two device classes, ***block*** and ***character***, and each class is associated with a group of device drivers. (For descriptions of the device classes, see "Block I/O System" and "Character I/O System" on page 3-14.) The major device number indicates which driver the device uses. The minor device number passes additional information to the driver (for example, to specify which of several identical printers should be used).

Device characteristics are recorded in ***special files*** (ordinarily located in the **/dev** directory). A process can have read and write access to a special file just as it does to an ordinary file. However, the read and write operations on special files produce device I/O. To perform I/O operations, a process simply has to access the appropriate special file; it does not have to deal with the characteristics of the device itself.

Block I/O System

A model block I/O device consists of randomly addressed memory blocks of a uniform size: 2048 bytes. The blocks are addressed 0, 1, . . . up to the size of the device (because the addresses start at 0, the highest-numbered address is the device size less one). A block device driver creates this model on a physical device.

Input and output for block devices is through a group of buffers. The system maintains a buffer for each block device. When a device issues a read request, the system searches the buffers for the requested block. If the block is in the buffer, the system makes it available to the device without any physical I/O. If the block is not in the buffer, the system

performs a physical I/O operation, replacing the least recently used block in the buffer with the requested block. The system handles block write operations similarly.

Generally, block (buffered) I/O increases system efficiency. First, block I/O can greatly reduce the number of read and write operations required to perform I/O. Second, block I/O allows the operating system to schedule I/O for the most efficient system operation. In some instances, however, block I/O is a disadvantage. If the system stops unexpectedly, there may be physically incomplete I/O—changes made to the buffers that were not yet written to the disk.

Note: Not all file I/O is done with the block I/O system. Files can be ***mapped*** directly into memory. A single segment of a mapped file can be shared by more than one program. Changes made to mapped files show up in the file immediately.

Character I/O System

The character I/O system includes all devices that do not fit the block I/O model. Unlike block I/O requests, character I/O requests go directly to the device driver.

A model character device places characters directly onto a queue until the queue contains the maximum number of characters. As soon as there are any characters in the queue, the I/O operation starts, removing the characters from the queue one at a time and sending them to the device. When the number of characters falls to a specific level, the process passing characters to the queue refills the queue to its maximum capacity.

However, many devices in the character class are more accurately described as ***raw*** devices. For example, I/O for both fixed-disks and streaming tapes occurs in units of bytes which have no relationship to characters. Raw devices are included in the character class only because they are not block I/O devices.

Using the Queueing System

A ***queue*** is an ordered list of jobs waiting to be done by the computer. The main job of the AIX queueing system is to manage printer use, especially on systems that have more than one printer. However, you also can use the queueing system to control access to other system resources, as is explained under "Friendly and Unfriendly Backends" on page 3-23. You can adapt the queueing system to your requirements easily. For example, by changing one word in a master configuration file, you can set a printer to take jobs in the order in which they are submitted or according to their sizes. Because your requirements of the queueing system may change periodically, you should understand how the system works and how to modify it—the subjects of this section.

Parts of the Queueing System

The queueing system has four parts:

The print program
: The **print** program accepts requests for print jobs and places them in the queue directory, **/usr/lpd/qdir**. (For more information on the **print** command, see *Using the AIX Operating System.*)

The qdaemon program
: A ***daemon*** is a process that starts when you start your computer and runs until you shut your computer down. The **qdaemon** is a daemon that keeps track of print job requests (or other output) and the printers (or other devices) available to handle them. The **qdaemon** process maintains queues of outstanding requests and sends them to the proper device at the proper time. The **qdaemon** also records printer usage data for system accounting purposes.

The backend program
: A ***backend*** program sends output to a particular device (a printer, for example). You do not run a backend program directly; rather, **qdaemon** runs a backend program, sending it the names of the files to be printed and any print flags that you specify. The backend returns this and other information to the **qdaemon** through a status file in the **/usr/lpd/stat** directory. You can use the **print -q** command to interpret and display this status information, such as how many pages were printed, what the status of the printer is, and how much of the print job is finished.

The configuration file
: The ***configuration file***, **/etc/qconfig**, describes the configuration of available queues and devices. Both the **print** command and the **qdaemon** process refer to the configuration file.

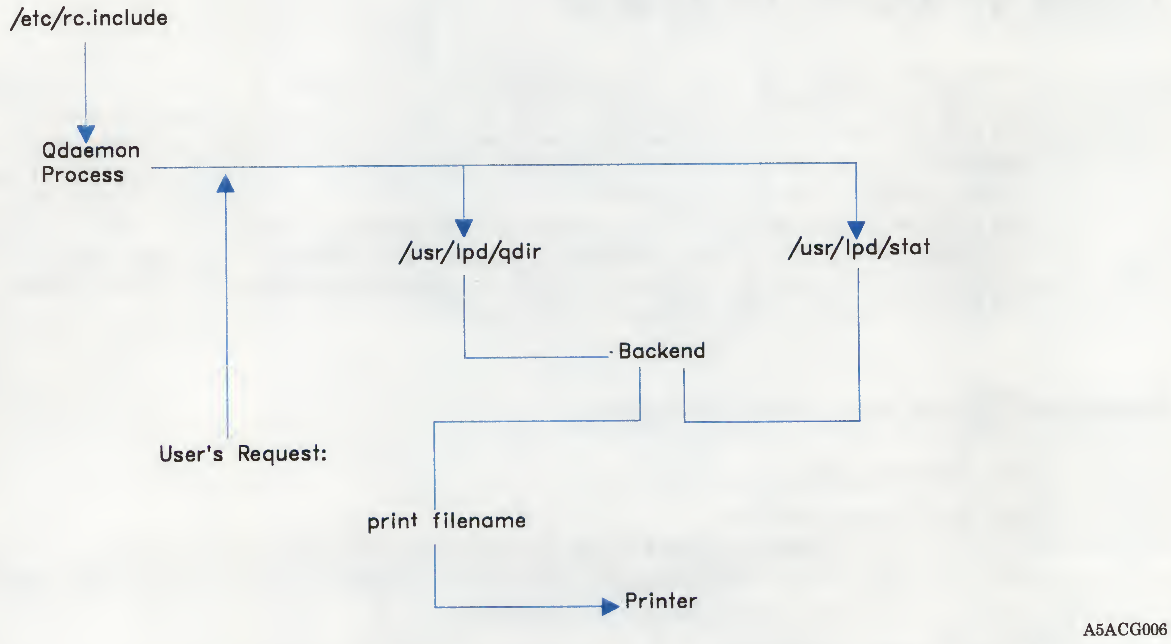


Figure 3-1. How the Queueing System Works

Queues and Devices

A ***queue*** is simply an ordered list of requests for a particular service. A ***device*** is something that can handle those requests one at a time. Each queue must be handled by at least one device, but it often may be handled by more than one.

Configuration

Often you do not have a choice about which device services which queue. For example, if your system has only one printer, then that printer must service the print queue. There are times, however, when you should consider carefully the relationship between queues and devices.

Note: If you have no printers and send something to print accidentally, problems can occur. The system will send the print job to the background shell and attempt to execute it there. For example, if your system has two identical printers located next to each other, you probably will want both printers to service a single printer queue.

You could accomplish this by placing the following stanzas in the configuration file **/etc/qconfig**:

```
lpr:
        argname = -l
        device = lpdev0, lpdev1

lpdev0:
        file = /dev/lp0
        backend = /usr/lpd/lp

lpdev1:
        file = /dev/lp1
        backend = /usr/lpd/lp
```

The first stanza describes the **lpr** queue. The `argname` line sets the flag used to request this queue, `-l`. A print command requesting this queue has the form `print -l` *filename*. The `device` line specifies which devices service the queue. The second and third stanzas describe the printers that service the queue. In each case, `file` is the special file associated with the printer, and `backend` is the file that contains the printer backend program. Once a print request reaches the top of the queue, the backend sends the file to the next available printer.

In another situation, you might want to define a different queue for each printer. For example, if the two identical printers are in different buildings, or if there is a difference in speed or quality between the two printers, you (or those who use your system) should be able to select a particular printer. In this case, the stanzas in **/etc/qconfig** could be:

```
lpa:
        argname = -la
        device = lp0

lp0:
        file = /dev/lp0
        backend = /usr/lpd/lp

lpb:
        argname = -lb
        device = lp1

lp1:
        file = /dev/lp1
        backend = /usr/lpd/lp
```

In this example, there are two queue stanzas, `lpa` and `lpb`. Following each queue stanza is a stanza describing the printer that services that queue.

Queue and Device Names

The names of queues and devices are similar, but there are important differences in how they work. Most system users do not use device names, since a **print** command is actually a request for a queue, not for a device.

Two lines in the **/etc/qconfig** file relate to the naming of queues:

- ***queue name***. This is the name of the stanza that describes the queue (like `lpa` and `lpb` in the previous example). The queueing system refers to a queue by its queue name and uses the queue name in any messages about the print job (including responses to the **print -q** command). The **qdaemon** takes queue names of the form **lp***xx*, so that it is possible to have more than 9 queues available.
- ***argname***. This is the name you use with the **print** command to request a specific queue (like `-la` and `-lb` in the previous example).

Although the `queue name` and `argname` can be the same, you may find it helpful to make them different so that, as in the previous example, the `queue name` can look like the name of a printer and `argname` can look like a command flag.

When more than one printer services a queue, you can request a particular printer by naming it explicitly. (Ordinarily, your request would be serviced by the first device on that queue that became available.) To name a device explicitly, use the `argname` of the queue serviced by that device, followed by a colon (:) and the `device number` of the printer. (Devices are numbered independently for each queue, starting at 0; device numbers refer to devices in the order in which they appear in **/etc/qconfig**.) The following stanza (taken from an earlier example) defines a queue service by two printers:

```
lpr:
        argname = -l
        device = lpdev0, lpdev1
```

The command **print -l:0** requests the printer `lpdev0`; the command **print -l:1** requests the printer `lpdev1`.

All references to devices by the **print** command and **qdaemon** are based upon the names contained in the **/etc/qconfig** file. Thus, within the guidelines given in "Changing the Configuration File" on page 3-25, you can choose your own names for queues and devices.

Status Control

By setting the status of queues or printers to `up` or `down`, you can control access to them. A status setting of `up` or `down` refers to how the system treats the queue or device, not to the physical state of any system component.

A queue is available to both the **print** command and the **qdaemon** if its stanza in **/etc/qconfig** contains the line:

```
up = TRUE
```

or if the stanza does not contain the line `up =` (since the default value for `up` is `TRUE`).

A queue is not available if its stanza contains the line:

```
up = FALSE
```

A queue that is up works normally. However, if a queue is down:

- The **qdaemon** command does not send jobs to devices on the queue.
- The **print** command does not accept requests for the queue.
- The **print -q** command shows that all devices on that queue are OFF.

The only way to set the status of a queue on or off is to change the **/etc/qconfig** file. Generally, you should not set the status of a queue to off unless you know that printers that service the queue cannot be used for several days (for example, if they are to be taken down for repair).

You can set the status of a device more easily, using the **print -dd** (device down) or **print -du** (device up) command. To set status down for the first device servicing queue l, use the command:

```
print -l:0 -dd
```

To set status up for the same device, use the command:

```
print -l:0 -du
```

In both examples, the device name -l:0 consists of the queue name (-l), a colon (:), and the device number (0).

The **qdaemon** does not send jobs to down devices, but **print** accepts requests for their queues. It also is possible for a backend to set the status of a device to off. Once a device is down, either because of a **print** command or the action of a backend, you must use the **print** command to set its status back to up.

The **qdaemon** records device status in a status file in the directory **/usr/lpd/stat**. When you start the system, **qdaemon** checks this status file before trying to run any device. Thus, you do not have to reset the status of devices each time you stop and restart your system.

Job Order

The ***discipline*** line in the **/etc/qconfig** file determines the order in which printers service the requests in a queue. In a queue stanza, the line:

```
discipline = fcfs
```

sets the order to *first come first served.* If there is no discipline line in the queue stanza, requests are serviced in first-come-first served (fcfs) order. The line:

```
discipline = sjn
```

in a queue stanza sets the order to *shortest job next.*

Each print job also has a ***priority number*** that can be changed with the **print** command flag **-pr**=*n*. Print jobs with higher priority numbers (*n*) are handled before requests with lower priority numbers. Any system user can alter the priority of a print request with a command such as:

```
print -pr=20
```

The default priority number is 15. The maximum priority number for ordinary users is 20. The maximum priority number for privileged users (su and members of the system group) is 30.

Spooling

You can use the **-cp** flag with the **print** command to create spool files. A spool file contains output information to be processed at a later date. The directory of the print file and the IDs attached to it determine those of the spooled file. By default the **qdaemon** leaves the directory of the spool file to **/usr/spool/qdaemon**. Both the user ID and group ID remain that of the print file.

The keyword **CD** in the **/etc/config** file determines how the **qdaemon** handles spool files.

```
lpr:
        argname = -l
        device = lpdev0, lpdev1
        CD = FALSE
```

In the above example, a **CD** of FALSE will cause the **qdaemon** to handle spool files in the default manner.

Note: If the keyword is omitted from the **/etc/config** file default handling will also result.

It is possible to send output to a spool file on a remote system. If you intend to spool output to remote systems, it is recommended you set the keyword **CD** equal to TRUE. A value of TRUE causes the **qdaemon** to change directory to the spool file directory and to translate the user and group IDs. This value allows the remote system access to the spool file regardless of the state of the server. For information on working with remote printer queues and translation of IDs, see "Using Distributed Services to Provide Remote Queues" on page 11-45.

Accounting

Accounting information consists of the user number, user name, and number of pages printed for each request. The backend program determines what constitutes a page. You can cause the queueing system to produce accounting records by placing the following line in the appropriate queue stanza in **/etc/qconfig**:

```
acctfile = /usr/adm/qacct
```

If a queue stanza does not contain an `acctfile` line or contains the line:

```
acctfile = FALSE
```

the queueing system does not produce accounting records for that queue.

The **qdaemon**, which produces the accounting records, does not create the accounting file. Thus, you can substitute another file name for **/usr/adm/qacct** if you choose.

You also can use a separate file to record the accounting data for each queue. Using separate accounting files can help you collect usage statistics for each queue (a useful ability if, for example, you are looking for ways to improve the efficiency of your queueing system) or allow you to assess different charges to the users of your system for the use of different printers.

For a full discussion of AIX facilities for accounting and monitoring system activity, see "Running System Accounting" on page 5-4 and "Using the System Activity Package" on page 5-24.

Backends

The `backend` line of the stanza for a device in the **/etc/qconfig** file determines what backend the **qdaemon** runs for that device. A line like:

```
backend = /usr/lpd/lp
```

specifies the name of the backend program and includes any flags that are to be passed to that program. A line such as:

```
file = /dev/lp
```

causes **qdaemon** to send the backend's standard output to this file (in this case, the special file associated with a printer).

The line:

```
access = both
```

gives the backend both `read` and `write` access to the file. The line:

```
access = write
```

or the absence of an `access` line in the stanza gives the backend only `write` access to the file. If `file` does not have a value, **qdaemon** ignores the value of `access`.

The default value for `file` is `FALSE` and is only used by backends that access devices without help from **qdaemon**.

Friendly and Unfriendly Backends

Ordinarily, the actions of the queueing system and a backend (as described so far) are neatly integrated. As long as a backend program follows certain conventions for communicating with **qdaemon** and the **print** command, it is called a ***friendly backend***. The queueing system also can handle ***unfriendly backends***—backends that do not follow these communication conventions. An unfriendly backend can be virtually any program; **qdaemon** requires no special communication or coordination with an unfriendly backend. The `friend` line in a queue stanza:

```
friend = TRUE
```

or

```
friend = FALSE
```

indicates whether a backend is friendly or unfriendly. If there is not a `friend` line in the stanza, the default value is `TRUE`.

You can use the queueing system to provide organized access not only to system devices (often printers), but to other system resources as well. These applications of the queueing system typically use unfriendly backends. For example, on a system that has a log file (**/etc/logfile**) available to all users, only one user at a time should be allowed to edit the file. By adding the following stanzas to **/etc/qconfig**, you can require all access of the file to be through the queueing system (that is, one user at a time):

```
log:
        argname = -lf
        device = logdev
        friend = FALSE

logdev:
        file = /etc/logfile
        backend = /bin/cat
```

To add a paragraph to the logfile, you now must first write the paragraph into a file (named, for example, `temp`) and then use the **print** command:

```
print -lf temp
```

When the queueing system services your request for access to **/etc/logfile**, the **print** command adds the contents of `temp` to **/etc/logfile**. The (unfriendly) backend in this example is a standard command (**cat**). The **qdaemon** writes the contents of the temporary file to the end of **/etc/logfile**, not the beginning. Any system users who do not have `write` permission to **/etc/logfile** can change the file only by going through the queueing system.

You also can use the queueing system to require queued access to a program. For example, suppose a program on your system (`/bin/bigjob`) requires so much processing that, if two people run it at once, your entire system slows down seriously. You can make

the program accessible only through the queueing system by adding the following stanzas to **/etc/qconfig**:

```
hog:
        argname = -b
        friend = FALSE
        device = bigjob

bigjob
        backend = /bin/bigjob
```

To run `bigjob`, enter `print -b` *name* (*name* is the name associated with the queued request and is used when you cancel the job, change its priority, or display its status, and is not necessarily the name of a file). The queueing system services requests for the `bigjob` program one at a time. Queued access is not appropriate for interactive programs.

Burst Pages

The term ***burst pages*** refers to printer output formatted so that each sheet of paper contains one page of output (on continuous form paper, the pages can be *burst* apart at the perforations). With the appropriate instructions in **/etc/qconfig**, most friendly backends print burst pages on their devices. For example, the following lines usually appear in the device stanza for a line printer:

```
header = always
trailer = group
feed = 3
```

The first line tells the backend to print a header page that gives the name of the job, who requested it, its date, and other information before every file printed. Other possibilities for the `header` line are:

```
header = group
```

which requests a header before every group of files for a single user, and:

```
header = never
```

(or the absence of a `header` line) which eliminates headers completely.

The `trailer` line in the device stanza tells the backend to print a trailer page, which gives the name of the user, only after a group of files is printed. (You also can use the `group` and `never` values with trailers.)

The `feed` line specifies that three `feed` (blank) pages are to be printed whenever the printer becomes idle. The feed operation pushes paper out of the printer so that it is easier to tear off. If the feed line in the stanza is:

```
feed = never
```

the backend is not invoked when the machine becomes idle, and therefore feed pages are not printed. The result of `feed = never` is usually the same as the result of:

```
feed = 0
```

except that the `0` value invokes the backend, but tells it not to feed any pages. For some printers, `feed = never` and `feed = 0` produce different results. For example, some line printers make a loud noise if the paper fold rests under the spinning drum. The backend for such a printer might interpret `feed = 0` to mean that three blank lines should be fed, moving the fold off of the printer drum.

The `align` value is another way to control the number of extra pages between print jobs. The following line in the device stanza tells the backend to print an aligning form feed before any job requested after the printer is idle:

```
align = TRUE
```

This value applies only to printers using continuous form paper. As a rule, alignment on such printers is unnecessary, since backends maintain proper paper alignment. However, on some continuous form printers, it is possible for users to get the paper out of alignment when they remove their jobs. If this is a problem on your system, the alignment line will help. The absence of an `align` line is the equivalent of the line:

```
align = FALSE
```

Changing the Configuration File

Both **print** and **qdaemon** read **/etc/qconfig** when they start. **qdaemon** starts when you start the system; **print** starts each time someone requests a print job. Thus, if you change **/etc/qconfig**, **print** will read the new version of the configuration file the next time it runs, while **qdaemon** continues to rely on the original version of **/etc/qconfig**. To avoid problems due to this inconsistency, give the command:

```
print -rr
```

which causes **qdaemon** to reread **/etc/qconfig** and reinitialize itself, based on the new version of **/etc/qconfig**, after all of its running jobs complete. (You also can cause **qdaemon** to reread the configuration file by shutting down and restarting the system, but the **print -rr** command usually is more convenient.)

You should be alert to two common problems associated with installing a new **/etc/qconfig**:

- If you install a new queue, someone may request that queue before you reinitialize **qdaemon**.
- If you remove an old queue, someone may request that queue at the last minute and find that it is no longer valid.

In either case, **qdaemon** logs an error message. You must determine whether the message refers to a new queue (in which case there is no longer a problem once you reinitialize **qdaemon**) or to an old queue (in which case the problem will exist until you remove the obsolete queue entries from **/usr/lpd/qdir**).

Keeping the qdaemon Running

Under normal circumstances, **qdaemon** starts when the system starts, runs until the system shuts down, and should require no attention from you. Under unusual circumstances, however, the **qdaemon** may stop running or be unable to perform its function. This section explains what you need to do under these conditions.

Any of the following conditions indicates that the **qdaemon** needs maintenance:

- **print** requests return the error message:

  `cannot awaken qdaemon (request accepted anyway)`

- **qdaemon** detects serious inconsistencies within itself and displays an error message
- **ps -ef** (the process status command that gives a full listing of all processes) does not show a process named **/etc/qdaemon** or **qdaemon**.

To restart the **qdaemon**, use the following command:

```
/etc/qdaemon
```

Generally, only privileged users can use this command. The new **qdaemon** goes through an initialization process.

If the **qdaemon** does not stay running, make sure that both **qdaemon** and **print** have the appropriate permissions. The user root owns both **qdaemon** and **print**. However, **qdaemon** and **print** must run as if they are owned by the user who starts them. The permission **s** sets the effective owner (user ID) of a process to that of the command that is running it. Thus, the appropriate permissions for these two commands are:

qdaemon -r-sr-sr--

To check these permissions, enter `ls -l /etc/qdaemon`.

If the permissions need to be reset, enter: `chmod 6554 /etc/qdaemon`. (You must have superuser authority to reset these permissions.)

print -r-sr-sr-x

To check these permissions, enter `ls -l /bin/print`.

If the permissions need to be reset, enter: `chmod 6555 /bin/print`. (You must have superuser privileges to reset these permissions.)

If you continue to have problems with **qdaemon**, you can reinitialize the entire queueing system by going through the following procedure:

1. If the **qdaemon** is running (use the **ps -ef** command to find out), end it with the **kill** command (**kill** *process id number*).
2. If any backends are running, use the **kill** command to stop them.
3. Delete the contents of the following directories:
   - **/usr/lpd/stat**
   - **/usr/lpd/qdir**
   - **/tmp/copies**
4. Restart the **qdaemon** with the command `/etc/qdaemon`.

Handling System Errors

Whenever some part of your AIX system does not function properly, the system has an ***error condition***. Some error conditions are minor as when, for example, you make a typing mistake and enter a command that the system does not recognize. Other error conditions can be so severe that they cause your system to halt. The first part of this section summarizes what you should do if your system halts unexpectedly. The remainder of this section describes the AIX facilities for collecting, analyzing, and reporting system errors:

- Error logging services
- The **dump** command
- The **trace** command.

Recovering from Unexpected System Halts

Under normal conditions, before you turn off the power to the system, you use the **shutdown** command to bring all system processes to an orderly halt. Among other things, the **shutdown** command assures that all files are updated and stops any running system processes. In the event of a power failure, or if someone turns off the power to the system without running **shutdown**, your file system may have inconsistencies. You may be able to restore your system to normal operation by simply supplying power again. As part of the normal initialization procedure, the system runs the **fsck** program. If the file system sustained only minor damage, the form of **fsck** that runs automatically may be adequate to repair the damage. If not, a system message will indicate what action to take next.

The / (root) file system can become too full to use. For protection, the system moves **/etc/passwd** to **/etc/opasswd** and **/etc/group** to **/etc/ogroup**. In this state, the system either does not accept a valid password or prompts you for a password when one should not be required. In either case, you cannot use the system. To correct the problem, first start the maintenance system and verify that the names of **passwd** and **group** have been changed. Then either delete a substantial number of files from the / file system or back up the / file system, create a larger minidisk for it, and restore it. Next, move **/etc/opasswd** and **/etc/ogroup** back to their original names. Finally, start the system, obtain superuser authority, and run the **df** command to verify that the / file system is large enough (usage should be about 90 percent or less).

If a system halt is not caused by power failure or a full / file system, you should see *Problem Determination Guide* to determine its cause.

Error Logging, Analysis, and Reporting

The following components provide the AIX error logging services:

event log
: Two files, **/usr/adm/ras/errfile.0** and **/usr/adm/ras/errfile.1**, of a predetermined size, which contain entries about system errors and other system events. When one file is full, the system begins to log events in the second file. When the second file is full, the system begins to overwrite data in the first file with new event entries.

error analysis facility
: A program that provides information about the probable cause of errors.

errdemon
: The ***error daemon process***, which collects error records from a buffer in memory, calls the error analysis facility, and writes error records, together with analysis information, to the event log. When it writes certain types of error records to the event log, the error daemon sends an alert message to the display.

error device driver
: The device driver, **/dev/err**, used by **errdemon** to write error records to the event log.

errpt
: A command that produces a report of logged errors from the data stored in the event log. **errpt** can produce summary or detailed reports, and you can specify what type of error records the report includes or the time span that the report covers. Generally, this is the only part of the error logging system that you work with directly.

When you start the AIX system, the error logging services start automatically. When you stop your system, the error logging services stop automatically.

To generate a report of entries in the event log, including errors, enter a command of the form:

`errpt` *flags filenames*

where *filenames* specifies the event log files. The **errpt** command accepts the following flags:

-s*date*	Ignores all records logged earlier than *date*. *date* must be in the form *mmddhhmmyy*.
-e*date*	Ignores all records logged later than *date*.
-a	Produces a detailed report on a single record.
-n*nodename*	Includes only error entries from *nodename* in the report.

-v*vmid* Includes only error entries from this system name (*vmid*) in the report.

-d*list* Limits the report to the types of error records specified in *list*. For the values that can appear in *list*, see **errpt** in *AIX Operating System Commands Reference*.

For more information about using **errpt** to analyze error data in the event log, see *Problem Determination Guide*.

There are two ways to limit the information in the error log report to that which you find useful:

- Periodically remove the error log files (you must have superuser authority):
  1. Enter `errstop` to stop the error daemon.
  2. Enter `del /usr/adm/ras/errfile.0`
  3. Enter `del /usr/adm/ras/errfile.1`
  4. Enter `/usr/lib/errdemon` to restart the error daemon.
- Narrow the error log report by selecting the **errpt** options to limit the error report generator to the type of information you find useful.

Memory Dump Services

The RT system has three facilities for recording its state at the time of a failure:

- VRM dump
- kernel dump
- full image dump.

The data collected by any of these **dump** facilities is intended to help you or the person servicing your system determine the cause of the failure.

The Virtual Resource Manager Dump

If the system detects a failure, it starts the Virtual Resource Manager dump automatically. If the system, or part of the system, appears to be hung up (but a dump has not been started automatically), you can start the Virtual Resource Manager dump by pressing **Ctrl-(Left)Alt-numpad8**.

Note: The message `Starting kernel dump....` displays when a dump occurs, regardless of whether a dump is started automatically or by pressing **Ctrl-(Left)Alt-numpad8**.

The Virtual Resource Manager dump program records its data on a diskette. You can use any formatted high-capacity diskette as a ***dump diskette***. However, since the system may not be able to format a diskette at the time you need one to hold dump data, you always should keep a dump diskette ready to use—store a dump diskette in the sleeve designed for that purpose in the back of *Problem Determination Guide.*

Besides the **dump** program, the AIX system provides a dump ***formatter***. The formatter, **dumpfmt**, processes the data collected by **dump**, producing information in a readable form. For more information about Virtual Resource Manager dump facilities, see *AIX Operating System Commands Reference* and *Problem Determination Guide.*

The Kernel Dump

If the system detects an operating system failure, it may start an operating system dump automatically. If the system, or part of the system, appears to be hung up (but a dump has not been started automatically), you can start an operating system dump by pressing **DUMP (Ctrl-(Left)Alt-End)**.

The operating system dump program writes its output to the dump minidisk on the fixed disk. To be used in problem determination, the data produced by this program must be copied to diskettes. See *Problem Determination Guide* for information about copying an operating system dump to diskettes.

Note: The **crash** command is a utility program designed to examine operating system dumps. If you are a very experienced computer user, you may find the **crash** command useful. See **crash** in *AIX Operating System Commands Reference.*

The Full Image Dump

A full image dump writes all of your system's real memory to diskettes (up to four). Generally, you should not make a full image dump unless you have been directed to do so by the person servicing your system. A full image dump takes longer to complete than a Virtual Resource Manager dump does, and you cannot use the dump formatter to inspect the data produced by a full image dump.

To be prepared to make a full image dump, you should have enough formatted diskettes available for your system:

System Memory	Number of Diskettes
2M bytes	2
3M bytes	3
4M bytes	4
5M bytes	5

System Memory	Number of Diskettes
6M bytes	6
7M bytes	7
8M bytes	8
8M bytes	8
9M bytes	9

Note: Every M byte requires 1 diskette to make a full image dump. So, a system as large as 16M bytes would require sixteen diskettes.

trace Services

The **trace** command monitors system events while a program runs. You can use **trace** data to analyze system performance and to detect problems with programs. The **trcrpt** command processes the data collected by **trace** into a usable form and writes it into a file.

The **trace** command monitors the system events specified in a ***trace profile***. The default profile is the **/etc/trcprofile** file. To change the events that **trace** monitors, you can either modify **/etc/profile** (with an editor) or create alternate **trace** profiles.

To use **trace**, enter `trace`, then start the program with the activity you want to trace. To stop **trace**, enter `trcstop`.

Generating a New Kernel

Customizing is the process of adapting the AIX for a particular set of requirements. When you customize a system, you define for the kernel such things as:

- Devices attached to the system (for example, printers, display stations, and fixed disks)
- Limits on system resources (for example, the maximum number of processes that can run at the same time or the maximum number of file systems that can be used at the same time).

The system is initially customized when it is installed; much of the customizing happens automatically during the installation process described in *Installing and Customizing the AIX Operating System.* You can customize some aspects of the system while it is in use (for example, with the **devices** command described in *Installing and Customizing the AIX Operating System*).

However, some system modifications require you to generate a new kernel. For example, if you modify ***system parameters*** or add new class of device drivers to your system, you must generate a new kernel. This section describes the procedure for generating a new kernel and provides general information about system parameters. It is beyond the scope of this section to discuss the potential advantages and disadvantages of modifying specific system parameters on your system.

Note: Generating a new kernel is not a routine procedure. You may never need to perform it.

Using the make Command

Two system description files, **/etc/master** and **/etc/system**, define the characteristics of your system:

/etc/master Contains system parameters and the stanzas that describe device drivers included in the system (whether they are in use or not). You can think of this file as a model.

/etc/system Contains entries for devices that make up the current system.

Depending upon the changes you want to make to your system, you may need to modify one or both of these files. For information about the types of changes you can make to the system description files, see "System Parameters" on page 3-35. Except for modifying the system description files, the process of generating a new kernel is automated.

To Generate a New Kernel

1. Use an editor to modify **/etc/master** or **/etc/system** as necessary.
2. Enter `cd /usr/sys`
3. Enter `make` and wait until the **make** command completes.
4. Test the new kernel. Verify changes to the new kernel before moving to the next step.
5. Enter `mv /unix` */unix.old* (where *unix.old* is a unique name for the old kernel).

   Note: To avoid confusion, it is recommend that you use a naming convention such that *unixyyddd.nnn* is the name of the old kernel. Where *yy* is the year, *ddd* is the day of the year, and *nnn* is the number of the backup.
6. Enter `mv unix.std /unix`

After you modify the system description file or files, change directories to **/usr/sys**. **/usr/sys** contains the **Makefile** file used by the **make** command to generate the new kernel. The **make** command, using the information in **Makefile**, generates the new kernel in two steps:

1. Uses the **/etc/config** command to translate **/etc/master** and **/etc/system** into a C language program **(conf.c)**.
2. Compiles the C program (with the **cc** compiler) and combines the compiled program with standard parts of the operating system to produce the new kernel, a file named **unix.std**.

Even if you make only a minor change to a system parameter, you should first back up your file systems (see "Backing up Files and File Systems" on page 2-45) and then test the new kernel. To test the new kernel, start the standalone shell (as is explained under "Running the Maintenance System" on page 2-5) and then:

1. Save the old kernel under a different name, for example:

   `mv /unix /unix.88257.001`
2. Move the new kernel to **/unix**, for example:

   `mv /usr/sys/unix.std /unix`
3. Shutdown the system and then restart it.

Note: Every version of the system should have a unique name. As far as is practical (given the amount of storage available on your system), you should keep old versions of the kernel until they become obsolete.

Once the new kernel is loaded, try read, write, and other operations that should indicate whether the kernel works properly. It is beyond the scope of this guide to discuss the testing and refinement of new versions of the kernel except for one general guideline: **When you generate a new kernel, thoroughly test every change you make and the general operation of the kernel.**

If the new kernel does not work, you must use the AIX Operating System Installation/Maintenance diskette to remove the new kernel and change the name of the original kernel back to **/unix**:

1. Start the standalone shell.
2. Mount the fixed disk:

   ```
   mount /dev/hd0 /mnt
   ```
3. Rename the faulty kernel:

   ```
   /mnt/bin/mv /mnt/unix /mnt/unix.bad
   ```
4. Rename the original kernel:

   ```
   /mnt/bin/mv /mnt/unix.88257.001 /mnt/unix
   ```
5. Issue the **sync** command to insure system integrity.
6. Unmount the fixed disk:

   ```
   umount /dev/hd0
   ```
7. Exit the shell.
8. Restart the system.

System Parameters

To a large extent, system parameters determine the system's capacity and performance. This section discusses several of the parameters that are most commonly reset to correct system resource problems. To modify any of these parameters, you first make your changes to the **/etc/master** file and then generate a new kernel (as described under "Using the make Command" on page 3-33). The values for the following parameters on your system are listed in the file **/etc/master**.

kbuffers Almost all disk I/O goes through a group of buffers. To avoid extra I/O operations, the system uses the buffers to maintain copies of the disk blocks that are being used most frequently. As the size of the buffer group decreases, the amount of disk I/O required increases, reducing the effectiveness of the buffering system. At the same time, an excessively large set of buffers may not improve the effectiveness of the buffering system, but will prevent the system from using the memory allocated to

buffers for other purposes. The ***kbuffers*** parameter specifies the number of buffers used by a particular kernel.

Note: Each buffer is 2048 bytes. If you specify zero for the **kbuffers** parameter, the system uses the physical size of the installed memory to determine the number of buffers. If your system has less then 2 megabytes of memory installed, the system uses 75. If it has 2 megabytes or more of memory installed, the system uses 150. Generally, the best number to use is between 10 to 25 percent of the total system memory.

The number you choose may depend on how the system is being used. For example, if the system is used mostly for engineering graphics, you might choose fewer buffers; if the system is a multi-user system, you might choose more buffers.

procs The ***procs*** parameter specifies the maximum number of simultaneous processes the kernel supports. Once this limit is reached, no new processes can start. As long as the maximum number of processes is running, any attempt to start a new process causes the shell to display a message telling you to try again later.

maxprocs The ***maxprocs*** parameter specifies the maximum number of processes that can be run by any particular user. **maxprocs** controls the number of processes for each user in the same way that **procs** controls them for the kernel. The value for **maxprocs** on your system is listed in **/etc/master**.

mountab The ***mountab*** parameter specifies the maximum number of file systems that can be mounted at the same time. In general, the value of **mountab** should be one or two more than the number of file systems you expect to have mounted at the same time. Thus, if you normally mount four file systems, the value for **mountab** should be at least 5 or 6.

inodetab The system keeps copies of the i-nodes for all active files in a table in memory. The ***inodetab*** parameter sets the size of this table. An i-node is considered active if it is the i-node of an open file, if it is the current directory of any process, or if a file system is mounted on it. Once the table is full, no other files can be used, and almost all new system activity will be delayed until some processes complete, freeing up space in the i-node table.

On most systems, an allocation of five or six i-nodes for each process is adequate. If the value of **maxprocs** is 40, the value of **inodetab** usually should be about 250.

filetab The ***filetab*** parameter specifies the maximum number of files that can be open at the same time. If the limit of the file table is reached, attempts to open new files will fail until running processes close some of their files. The value of **filetab** should approximately equal the value of **inodetab**.

callouts Certain activities must be timed (that is, they require a predetermined interval between one action and another). Device drivers that use timed activities schedule them with the kernel's ***callout*** mechanism. The **callouts** parameter specifies how many entries are in a callouts table and thus, the maximum number of timed actions that can be scheduled (pending) concurrently. If the maximum number of callouts is reached, the system halts.

The callout mechanism is used most commonly by the display station device drivers. The value for **callouts** should be 25 to 100 percent more than the number of devices on the system.

texttab The AIX system makes it possible for all processes that are running the same program to share a single copy of the program (rather than loading a separate copy of the program into memory for each process). The ***texttab*** parameter determines the size of a text table. The text table contains one entry for each active shared text (program) segment in the system. If the text table fills up, no other text segments can be shared among processes until some processes complete and create some room in the text table. The value of **texttab** is usually 30 to 50 percent of the number of processes on the system.

This section does not discuss a number of the system parameters. The **/etc/master** file contains all of the system parameters. System parameters also are documented in *AIX Operating System Technical Reference*.

Chapter 4. Additional System Management Topics

CONTENTS

About This Chapter

This section covers an assortment of topics that you may find useful in managing the AIX system. If your system supports several users, you also should acquaint yourself with the AIX facilities for accounting and monitoring system activity. "Running System Accounting" on page 5-4 describes the AIX accounting facilities and how to use them. "Using the System Activity Package" on page 5-24 describes the facilities for monitoring system activity.

Automatically Installing AIX Operating System

If you are installing several systems, or you wish to automate the install process, you can create a **user_install** file on the AIX Operating System Installation/Maintenance diskette. The **user_install** file contains a set of responses to System Management menus. When you insert the AIX Operating System Installation/Maintenance diskette, the **maintshell** program checks to see if **user_install** exists. If the file exists, the System Management menus run automatically using the responses found in the file. After this process completes, the user is prompted to enter the proper program diskettes or tapes.

For users with several systems to install or update, the **user_install** file insures that all users will be operating under the same system configuration. Supervision of every installation is not necessary.

The **user_install** file resembles a shopping list. Each line in the file contains one response. The system responds to the file as if a user were actually entering responses. Thus, the file should contain only responses the system would normally expect.

Notes:

1. Do not put comment or blank lines (lines created by the **Tab** key or cursor movement) in the **user_install** file.
2. Responses include any key you normally enter. For instance, if the menu offers the option:

   ```
   To CANCEL and go back to the SYSTEM MANAGEMENT MENU,
   press F3.
   ```

   and you choose to cancel instead of selecting a menu option, your response would be `F3`.

Figure 4-1 on page 4-5 shows the structure of a **user_install** file for an installation using the default responses.

```
1
2
1
root
1
1


y


4
1
```

Figure 4-1. Default responses in a user–file

After each response remember to press **Enter** to move to the next option. The return character will not display in the file, but the installation program requires it. In Figure 4-1 the two blank lines between the `y` and `4` responses contain end of line indicators. When you press the **Return** key the system creates these markers.

Figure 4-2 on page 4-6 gives an example of a **user–install** file where the installation defaults are changed.

```
1
2
3
1
20000
5000
2
5500
13750
6
root
1
3


y


4
1
```

Figure 4-2. Non-default responses in a user_file

You can use the **user_install** file to create any installation or maintenance procedure that does not require direct user interaction, such as entering a diskette to complete a menu choice. For example, to create a fixed-disk minidisk, your **user_install** file would be similar to Figure 4-3.

```
2
3
1
1
1
2000


^[[003q4
1
```

Figure 4-3. Fixed disk responses in a user_file

Notes for creating user–install files:

1. Do not use the cursor keys to move from one line to another when creating the **user–install** file, using cursor keys or the **Tab** key in this way creates blank lines in the file.

   Each line should contain either a response to a menu and its accompanying carriage return character or only a carriage return. A carriage return character is created when you use the **Return** key and will not be visible in the file. Always use the **Return** key as if you were entering the items interactively with the menu.

2. The **vi** editor requires special consideration when creating a **user–install** file containing function key responses. For example, to enter the F3 response and choose menu option 4, you must enter the following:

   ```
   ^[[003q4
   ```

   The ^[[003q portion represents the code for the F3 key. It is entered using the **vi** subcommand **Ctrl-V**. The 4 represents the response immediately following the F3 key. This response must appear on the same line as the F3 sequence. For information on the **vi** editor see AIX Operating System Commands Reference.

3. The **user–install** file must contain exact responses to the menu items. If the responses are not in the correct order or if a response is excluded, the installation will not complete or will complete incorrectly.

Preparing the Automatic Install diskette

1. Install or update at least one system by working through the menus interactively.
2. As you work through the menus, make notes of your responses. Remember to record them in the order you enter them.
3. Make a working copy of the AIX Operating System Installation/Maintenance diskette.
   a. Insert the AIX Operating System Installation/Maintenance diskette into drive 0 (or A).
   b. On the command line enter the following:

      ```
      dd if=/dev/fd0 of=INSTALL bs=8192
      ```

      This copies it onto your system.

      Note: If you have two 1.2M disk drives enter `dd if=/dev/fd0 of=/dev/fd1` and go to step 4 on page 4-8.
   c. Remove the AIX Operating System Installation/Maintenance diskette.
   d. Insert a formatted diskette into drive 0 (or A).
   e. On the command line enter the following:

      ```
      dd if=INSTALL of=/dev/fd0 bs=8192
      ```

4. Mount the working copy of the diskette using the **mount** command.
5. Change directory to the **etc** directory on the working copy diskette. For example, if you mounted over directory **/diskete0**, you change the directory by issuing `cd /diskette0/etc`. Then ensure that the **maintshell** file is on your diskette using the **ls** command.
6. Using an editor, create a **user_install** file.
7. Unmount the working copy.
8. Insert the AIX Operating System Installation/Maintenance diskette working copy in drive 0 (or A) of a system to be installed and IPL the system.

   The system will run through the menus and then prompt you to install the proper program diskettes.
9. Ensure that your **user_install** file executed the installation correctly before going on to install other systems.

   Note: At this point, if you have several installations to complete you may want to make additional copies of the working copy.

For information on formatting diskettes, see "Formatting Diskettes" on page 2-43. For information on mounting diskettes, see "Mounting and Unmounting Diskette File Systems" on page 2-43.

Updating the System and Installing Local Programs

There are three ways in which you can alter the basic features of the RT system:

- Install additional licensed programs or components of licensed programs (using the **installp** command as is explained in *Installing and Customizing the AIX Operating System*).
- Make updates to one or more licensed programs or components of licensed programs that are already installed.
- Create your own programs.

This section contains guidelines for updating your system and installing your own programs.

Updating the System

An ***update*** is an improvement for some part of the system. An update may apply to one or more licensed programs or components of licensed programs. If IBM supplies updates for your AIX system, you will receive a diskette containing the update. You will use the **updatep** command to apply updates.

Updates for IBM provided software products on your AIX system are packaged together on the same diskette. You select the products for which you want to apply updates. After the updates are applied, you can selectively ***commit*** or ***reject*** the complete set of updates for each product based on the results of a test period. You must be a member of the **system** group to run the **updatep** command.

You start an update by running the **updatep** command, which performs five functions:

updatep -a	Apply selected product updates from the diskette.
updatep -c	Commit selected product updates that have been applied.
updatep -r	Reject selected product updates that have been applied.
updatep -s	Present status about all currently applied product updates.
updatep -A	Lists the version numbers of the licensed program applied to your system with **updatep**.

You must enter the appropriate action when you enter the **updatep** command.

A menu is presented if you choose one of the first three actions. If you choose **apply**, the menu contains every product on the diskette for which there is an earlier version on the system (that is, a version with a lower update level). If you choose either **commit** or **reject**, the menu contains all of the products on the system that have applied updates.

Select the appropriate products from the **apply**, **commit**, or **reject** menu. The action (**reject**, for example) is performed for all selected products. You can use **updatep -A** to view all the installed licensed program versions, version numbers, and apars associated with each version on your system. Figure 4-4 shows the type of information you receive with **updatep -A**:

```
Ined   Version 2.2.1
  IX00975 INed - Underline does not work on vt100
  IX01005 INed - Del Key vt220 does not work

tcpip Version 2.2.1
  IX00995 tcpip - tn to vax hangs virtual terminal
  IX01115 tcpip - . .
```

Figure 4-4. Displaying apars with updatep

If you would like to view history of a particular licensed program, specify **updatep -A** *programname*. You can specify more than one licensed program.

Note: The **updatep -A** program will only display information on programs installed with **updatep.**

For additional information on using **updatep**, see **updatep** in *AIX Operating System Commands Reference.*

Installing Applications and Local Commands

If you create your own commands or programs (shell programs, for example), the following guidelines will make them convenient to use:

- **Create a directory for local commands**. If you create commands that are to be available to all system users, place them in a single directory, for example, `/u/local`. If you create commands that are to be available only to members of certain groups, create a different command directory (for example, **/u/local/system**) and set its permissions appropriately.

By keeping local commands together in one (or two) directories, you make it very simple to:

- Remember the path name of the command.
- Determine what local commands are on the system (by listing the contents of the local command directory).

- **Create a directory for private commands**. If you (or other system users) create commands for private use, keep them in a single subdirectory of the **$HOME** directory (for example, **/u/tom/bin**). This type of organization has the same advantages for an individual user that it does, on the larger scale, for all users of the system—it is easy to remember where the commands are and thus, to distinguish the standard system from the parts of the system that you have added or created.

Setting the System Date

The RT system has an internal, battery-powered clock. You set the time and date when you install the system and the clock maintains the time and date whether power to the system is on or off. Should you need to reset the time or date (for example, to synchronize system time with standard time or because of a battery failure) you can do so with the **date** command. You also can use the **date** command to find out what the current system date and time are.

To set the date and time, enter a command of the form:

`date` *mmddhhmm.ssyy*

where:

mm designates the month.
dd designates the day.
hh designates the hour, using a 24-hour clock (for example, 7 p.m. is represented as 19).
mm designates the minutes.
.ss designates the seconds.
yy designates the year.

Each time or date designation must be two digits long (for example, the date designation for July is 07). For values that do not need to be changed, type spaces (for example, if the system's current month designation is correct, but you need to change the setting for day, type two spaces and then the new date).

Warning: Do not set the date or time while other users are logged in to the system.

You must have superuser authority to set the time and date.

In the following example, the **date** command sets the date and time to April 18, 3:15.32 (32 seconds after 3:15) p.m., 1985:

```
# date 04181515.3285
#
```

To display the system's current date and time values, simply enter: `date`.

For more information about the **date** command, see **date** in *AIX Operating System Commands Reference.*

Running Commands at Pre-set Times

The **cron** command resembles a clock that can run shell commands at pre-set dates and times. You do not enter the **cron** command; rather, **cron** is included in the **/etc/rc** file and starts during the system initialization process. Specify what commands are to run and when they are to run, with either the **at** command or the **crontab** command:

at Use **at** when a command is to be run only once.

crontab Use **crontab** when a command is to be run regularly.

Using the at Command

The **at** command takes its list of commands from standard input. Standard output and standard error from the commands are mailed to the user who ran the **at** command unless they are redirected.

To Use the at Command

1. Enter:

   `at` *time date +increment*

2. Enter the commands to be run (either one command per line, or separated by **;** [semicolons]).
3. Press **END OF FILE**.

Following is an explanation of each **at** command parameter and how to set it:

time Required parameter. Specify *time* in one, two, or four digits. **at** interprets one- and two-digit numbers as hours, four-digit numbers as an hour and minutes (for example, **10** means 10 o'clock, **1043** means 10:43). Though not required, you can separate the hours and minutes with a colon (**10:43**).

To indicate whether the time is a.m. or p.m., do one of the following:

- Append an `am` or `pm` suffix to *time* (for example, `10:43pm`).

- Use a 24-hour clock (for example, 10:43 p.m. is `22:43`). If you do not supply an `am` or `pm` suffix, **at** interprets the time based on a 24-hour clock.

You also can use the following special names in the **at** *time* specification:

noon
midnight
now
next

date Optional parameter. Specify *date* in one of the following ways:

- *monthname daynumber,yearnumber*. For *monthname*, use a three-letter abbreviation (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec). For *daynumber*, use a one- or two-digit number. The *yearnumber* is an optional two-digit number preceded by a comma.
- *dayname*. Specify the name of a day of the week, either fully spelled out or abbreviated to three letters.

The **at** command also recognizes the following special days:

today
tomorrow

If you do not specify *date*, **at** interprets the time to be:

`today` if *time* is later than the current time
`tomorrow` if *time* is earlier than the current time.

+increment Optional parameter. The *+increment* parameter, which is sometimes a more convenient way to specify time or date, is a number followed by one of these special words:

minute or **minutes**
hour or **hours**
day or **days**
week or **weeks**
month or **months**
year or **years**.

For example, to run a command at this same time tomorrow (rather than specifying the exact time) you could set up an **at** job like the one in the following example:

```
$ at now + 1 day
skulker
END OF FILE  .
$ _
```

Enter `at -l` to list the **at** jobs you have scheduled. Note that `at -l` lists **at** jobs by the numbers **at** assigns them. To see what commands a job contains, look at the file /usr/spool/cron/atjobs/*jobnumber*.

To cancel a scheduled command, use the **-r** flag:

`at -r` *jobnumber*

Using the crontab Command

The **crontab** command copies a specified file into the directory that contains all **crontab** files. The **cron** command (started by **/etc/rc** at system initialization) runs commands according to instructions in **crontab** files.

The general format of the **crontab** command is:

`crontab` *filename*

where *filename* is the name (or path name) of a file that contains information **cron** uses to schedule commands.

A **crontab** file can contain more than one entry (one entry corresponds to one command or command group). Each entry is on a separate line and consists of the following six fields, separated by spaces:

minute hour day/month month/year day/week command

Each field except *command* can contain:

- A number in the specified range

  Minute (0-59)
  Hour (0-23)
  Day of the month (1-31)
  Month of the year (1-12)
  Day of the week (0-6 for Sunday-Saturday)

- Two numbers separated by a - (hyphen) to indicate a range of values (for example, `4-15`)
- A list of numbers separated by commas, which specifies each number in the list (for example, `4, 8, 17`)
- An * (asterisk), which specifies all possible values.

cron runs the command at the specified time. If you include a % (percent sign) in the sixth field, **cron** takes everything before the % as the command and everything after it as input for the command.

Following is a sample **crontab** file, named `happy`, which you can create with an editor:

```
0 16 10-31 12 5 wall%HAPPY HOLIDAYS!
```

This **crontab** file writes a message to all logged-in users (**wall**) at 4:00 p.m. (0 minutes, 16 hours), from the 10th through the 31st (10-31) of December (12), on Fridays (5). The **wall** command takes the text following the % as its standard input.

Once you create `happy`, enter it into the **crontab** directory with the command: `crontab happy`.

You can modify the action of **crontab** with the following flags.

-l Lists the contents of your **crontab** file.

-r Removes your **crontab** file from the **crontab** directory.

Note: If you are logged in as the user **root**, you cannot run **crontab** on a file named **/usr/spool/cron/crontabs/su** unless **su** is the first entry in **/etc/passwd**. Similarly, if you are logged in as the user **su**, you cannot run **crontab** on a file named **/usr/spool/cron/crontabs/root** unless **root** is the first entry in **/etc/passwd.**

Using the skulker Command

The **skulker** command cleans up file systems by removing unwanted files. It is a shell script you can modify to meet your local needs. For instance, you can modify your system to invoke **skulker** daily. To create this modification, edit the following line:

```
#0 3 * * * /etc/skulker
```

in the **/usr/spool/cron/crontabs/root** file by removing the # (pound sign). The **crontab** command will now automatically run **skulker** on your system at 3:00 a.m. daily.

If your system is part of a Distributed Services or NFS network, it is recommended that you use **skulker** with caution because it does not distinguish between local and remote file systems when removing files. The **skulker** command running on these types of networks could delete remote files and increase network traffic.

You can modify **skulker** so it searches only the **/native** local file system by doing the following:

1. Edit the **/etc/skulker** shell script by removing the # (pound sign) from the line `#NATIVE=/native/`.

2. Find the line `NATIVE=/` and comment out the line by placing a # at the beginning of the line.

Before restricting **skulker** to the local file system check to see if the **/native** file system is mounted. If Distributed Services is installed on your system, the **/native** file system is established from **/etc/rc** when you start your system. The system mounts the root directory onto the **/native** directory before doing any remote mounts. The path to the root directory is not affected by the later remote mounts. The **skulker** command can access any of the local files through the **/native** file system.

If NFS is installed on your system but Distributed Services is not, the **/native** file system is established when you start the system only if you have edited the **/etc/rc** file. To edit the **/etc/rc** file, find the following group of lines in the file:

```
if [ -d /usr/lpp/ds ]
then
    mount -i / /native
    unmount /native/vrm
fi
```

Then, change `/usr/lpp/ds` to `/usr/lpp/nfs`.

Monitoring Files and Directories that Get Larger Automatically

Even though the fixed-disk on the RT system holds a large amount of data, it can become crowded, even full. Certain files on the system can grow automatically, using disk space to store data that are no longer useful. Following is a list of files that you may or may not have on your system. If you do have any of these files, you should check them periodically and remove all useless data.

- Accounting files:

  /usr/adm/wtmp

  /usr/adm/pacct

  /usr/adm/acct/nite/*

- Other files:

  /usr/adm/cron/log

  /usr/spool

  $HOME/mbox

Also check the **/tmp** directory for files that you can delete. Ordinarily processes should remove their temporary files from **/tmp**, but they may not always do so.

You can use the following shell procedure to locate files in specified directories (in this case, `/usr/adm`, `/usr/lib`, and `/usr/spool`) that are larger than a certain number of blocks (in this case, 50):

```
for i in /usr/adm /usr/lib /usr/spool
do
    find $i -size +50 -exec ls -l {} \;
done
```

Finding Files and Directories

The **find** command searches the file system for all file names that match a specified ***expression*** (characteristic or group of characteristics). For example, the following **find** command lists the complete path name of all files in the file system with the file name **.profile**:

```
find / -name .profile -print
```

The / starts the **find** command at the root directory; the search continues through all subdirectories of /. The `-name` parameter specifies the name that **find** is to match. The `-print` parameter causes **find** to display the path name of each file that matches the `-name` expression.

The `-name` expression is just one of several expressions that you can use with the **find** command. For more information about **find** and its expressions, see **find** in *AIX Operating System Commands Reference.*

You can use the **find** command together with the **skulker** and **xargs** commands to periodically remove unwanted files from the file system. For more information, see **skulker** and **xargs** in *AIX Operating System Commands Reference.*

Managing Display Station Features

This section covers two topics:

- How to automatically set the characteristics of any display station on your system
- How to use the special features of the main display station.

Setting Display Station Characteristics Automatically

A display station has standard characteristics, such as the length of the lines displayed on the screen, the number of lines per screen, and whether the special command line editing functions are available. However, you can use the **stty** command to change certain display station characteristics. (For more information about **stty**, see *Using the AIX Operating System* and **stty** in *AIX Operating System Commands Reference*.)

If you want the display station characteristics set in a particular (nonstandard) way every time you log in, you may find it convenient to have the system run **stty** automatically. Depending upon your requirements, you can do this in one of two ways:

- Place the **stty** command in the **/etc/profile** file.

  Each time you start the system, the system automatically runs the commands in **/etc/profile**. If you include **stty** in **/etc/profile**, every display station on the system starts with the same characteristics.

- Place the **stty** command in your user profile file, **$HOME/.profile**.

  Each time you log in, the system automatically runs the commands in **$HOME/.profile** (the individual user's profile). Commands in **$HOME/.profile** supersede those in **/etc/profile**. Therefore, if you include the **stty** command in your user profile, the display station always has the same characteristics when you first log in.

For more information about these two profiles, see "/etc/profile and $HOME/.profile" on page 2-34.

Managing Special Features of the Main Display Station

The **/dev/hft** device driver provides the special features of the main display station. (**hft** stands for high function terminal.) Another device driver, **/dev/console**, also can run the main display station (or ***console***), but it does not provide the special features).

Note: You should open another virtual terminal with `open sh` to run commands and programs; thus, leaving **/dev/console** free to receive error messages. To ensure that error messages do not scroll off the screen, enter `stty page length 22`.

"Managing the Virtual Terminal Feature" explains how to manage virtual terminals, a special feature you can use directly. "Additional Main Display Station Features" lists some other special features provided by the **/dev/hft** device driver.

Managing the Virtual Terminal Feature

As explained in *Using the AIX Operating System*, a virtual terminal is one of several equivalents of a display station, one or more of which can be available at the main display station concurrently. The **actman** (activity manager) command monitors virtual terminal activity.

To Enable actman

1. Enter:

   `users`

2. Use the **c** (change) subcommand to place `/bin/actman` in the `program` field for the appropriate user.
3. End **users**.

The **actman** command creates the initial shell and then monitors the number of open virtual terminals until all are closed. If you try to end the initial shell before all virtual terminals are closed, **actman** restarts the initial shell.

If you do not have virtual terminal capabilities (that is, you are not logged in at the main display station), the system replaces **actman** with the initial shell program.

Additional Main Display Station Features

In addition to virtual terminal capability, the device driver for the main RT display station provides several other special features:

- Sound control
- Keyboard control
- Locator (mouse) control
- Access to an extended character set
- Access to multiple fonts (character styles)

You generally cannot use the extended character set and multiple fonts directly. That is, there is no command or key to turn them on and off. Rather, these features are available, through system calls, to programs that require them. For more technical information about the **hft** device driver and the system calls associated with it, see *AIX Operating System Technical Reference.*

Controlling Sound

The **sound** command controls the volume of the sound output (the console bell and the keyboard click). You can set these two sound specifications independently of each other. There are two sets of flags for the **sound** command. The following flags control the volume of all sound output:

-h Sets the volume to high.

-l Sets the volume to low.

-m Sets the volume to medium.

-o Turns the volume off.

The second set of flags whether the click is produced:

-c Turns clicking on.

-q Turns clicking off.

The **sound** command in the following example sets the sound volume to low and turns the click function off:

```
$ sound -lq
_
```

Controlling the Keyboard

The **keyboard** command controls the ***delay rate*** and ***repetition rate*** of the keyboard. The delay rate is the interval from when you press a key to when it begins to repeat. The repetition rate is the number of times the key repeats per second. These rates are set at system startup to 500 milliseconds and 14 characters per second, respectively.

The **keyboard** has two flags:

-d*rate* Sets the delay rate to the specified value. *rate* can be **300**, **400**, **500**, or **600**.

-r*rate* Sets the rate of repetition to the specified value. This *rate* can be an integer from **2** to **40**.

The following **keyboard** command sets the delay to 300 milliseconds and the repetition rate to 40 characters per second:

```
keyboard -d300 -r40
```

Controlling the Locator (Mouse)

The **locator** command controls the rate at which the system checks the position of the locator (mouse). You can specify any of the following rates: **10**, **20**, **40**, **60**, **80**, or **100** times per second. At system startup, the locator rate is set at **60**.

The following **locator** command sets the rate at **40** times per second:

```
locator -r40
```

Changing the Physical Display

With the **display** command, you can change the physical display assigned to the current virtual terminal and the virtual terminal characteristics or assign a default display to be used when you open a virtual terminal. You can request only displays that are actually installed on your system.

The flags to the **display** command are:

-b	Changes the background color on the display.
-c	Changes current virtual terminal display.
-d	Sets default display to be used when a virtual terminal is opened.
-f	Changes the foreground color on the display.
-m	Changes the DMA pinned page size.
-p	Change the color palette to be used.
-t	Changes the font.

The **display** command with the **-c** and **-d** flags accept the following parameters:

pcmono	PC Monochrome Adapter and Display
egamono	Enhanced Graphics Adapter and PC Monochrome Display
egacol	Enhanced Graphics Adapter and Display
advmono	Advanced Monochrome Graphics Adapter and Display
advcol	Advanced Color Graphics Adapter and Display
extmono	Extended Monochrome Graphics Adapter and Display
megapel	IBM Megapel Display Adapter and IBM 5081 Display Models 16 and 19.

To see a menu of available options, enter **display** without flags or parameters. For example, when you enter `display -t`, you see a menu that contains a list of the available fonts. Choose one that is suitable for the work you will be doing. Not all fonts are available on all displays, nor do they all work with all programs.

The following **display** command changes the current virtual terminal display to an attached PC Monochrome Adapter and PC Monochrome Display:

```
display -c pcmono
```

You can use the **-c** and **-d** flags at the same time. The following **display** command changes the current virtual terminal display to the PC Monochrome Adapter and Display and makes the Enhanced Graphics Adapter the default display:

```
display -c pcmono -d egamono
```

For more information, see **display** in *AIX Operating System Commands Reference.*

TERM Values for Different Displays, Adapters, and Terminals

Change the value of `TERM` in the **.profile** file, and add `export TERM` to the **.profile** file.

Use the following values for `TERM`:

Display/Terminal	Adapter	Value
IBM Personal Computer Display	IBM Monochrome Display and Printer Adapter	ibm5151
IBM Personal Computer Display	IBM PC Enhanced Graphics Adapter	ibm5154
IBM Personal Computer Enhanced Color Display	IBM PC Enhanced Graphics Adapter	ibm5154
IBM RT Advanced Monochrome Graphics Display	IBM RT Advanced Monochrome Graphics Display Adapter	ibm6153
IBM Advanced Color Graphics Display	IBM Advanced Color Graphics Display Adapter	ibm6154
IBM Extended Monochrome Graphics Display	IBM Extended Monochrome Graphics Display Adapter	ibm6155

Figure 4-5 (Part 1 of 2). TERM Values for Different Displays, Adapters, and Terminals

Display/Terminal	Adapter	Value
IBM 3161 ASCII Display Station	n/a	ibm3161
IBM 3163 ASCII Display Station	n/a	ibm3161
DEC VT100[1] (terminal)	n/a	vt100
DEC VT220[1] (terminal)	n/a	vt220
IBM 3161 ASCII Display Station with cartridge[2]	n/a	ibm3161-C
IBM 3162 ASCII Display Station	n/a	ibm3161
IBM 3162 ASCII Display Station with cartridge[2]	n/a	ibm3162
IBM 5081 Display	IBM Megapel Display Adapter	ibm3162

Figure 4-5 (Part 2 of 2). TERM Values for Different Displays, Adapters, and Terminals

---

1 DEC, VT100, and VT220 are trademarks of Digital Equipment Corporation.

2 International character support

Managing Printers

There are two main management tasks associated with printers on the RT system:

- Adding new printers to the system.

  For information about adding a printer that is supported by the RT system, consult the documentation that comes with the printer and the section of *Installing and Customizing the AIX Operating System* that discusses the **devices** command.

  For information about adding a printer that is not supported, consult the documentation that comes with the printer.

- Altering the way printers work (the subject of this section).

When you print a file, the system sends a stream of codes to the printer. Some of the codes cause the printer to print particular characters (for example, a-z, A-Z, and 0-9). Other codes cause the printer to print characters or files in a particular way (for example, underscoring certain characters or making every page 60 lines long).

If you want to send different character codes to the printer (for example, to change the word `that` to the word `this`), you simply edit your file—you do not have to understand that codes are involved. However, altering the way a printer works is somewhat more complicated—you must understand what happens when you print a file, what options you have for sending control information to the printer, and what printer characteristics you can control.

The Printing Process

You use the **print** command to send a file to a printer. Often you may do nothing more than enter a command of the form `print` *filename* and wait for the printer to complete the job.

A file does not go directly to the printer, however. First, the **qdaemon** command places the print request in a queue. The print request stays in the queue until a printer becomes available, at which point **qdaemon** runs the **piobe** (printer input/output backend) command. (For more information on queues and the **qdaemon** command, see "Parts of the Queueing System" on page 3-15.)

The **piobe** command processes the file and sends it, along with control information, to the printer. Thus, the printer receives a data stream composed of the contents of the file and the control information supplied by **piobe**.

Controlling the Printing Process

You can add printer control information to the printer data stream in the following ways:

- Include printer control codes in the file.

  All printer control information that is unique to a file should be included in that file. For example, if you want to underscore the title of a book or print a paragraph in bold type, you must use an editor to insert codes into your file that cause the printer to start and stop the special treatment at the correct places. Appendix A, "Printer Control Codes" on page A-1 includes the RT printer control codes. Consult the documentation for your editor to learn how to use it to enter control codes into a file.

- Supply piobe flags with the print command.

  The **piobe** command recognizes a number of flags that control printer operations such as:

  - Starting and stopping condensed, emphasized, double-wide, and double-strike printing
  - Printing in specified colors
  - Setting the left, right, top, and bottom margins
  - Setting the number of lines per vertical inch
  - Maintaining the horizontal position on the print line for a line feed or vertical tab control.

  All of the **piobe** flags are listed and explained under **piobe** in *AIX Operating System Commands Reference.*

  If you want to use particular **piobe** flags for a single print job, specify them along with the **print** command. For example, the **piobe** flag for setting form (page) length is `-fl=`*num*, where *num* is a number of lines. If the standard **piobe** setting is `-fl=66` and you want the file `printtest` to be printed with 45 lines per page, you can enter the command:

  ```
  print -fl=45 printtest
  ```

  The flag on the command line overrides the standard **piobe** setting for this one job. However, the standard **piobe** form length setting remains 66.

- Change the standard piobe settings.

  To modify some printer characteristic for all print jobs, modify the standard **piobe** flags; **piobe** flags are contained in the **/etc/qconfig** file. (The **piobe** flags are listed under **piobe** in *AIX Operating System Commands Reference.*)

For example, if you want the standard page length for all print jobs to be 45 lines, you can use an editor to modify the **/etc/qconfig** file, setting the value of **-fl** to 45. Regardless of the page length set in **/etc/qconfig**, you always can specify a different page length for a particular print job by supplying a different value for **-fl** when you enter the **print** command.

Maintaining System Performance

System performance can be a highly specialized and complex subject, further complicated by the fact that a tactic that improves the performance of one system might actually impair the performance of a similar system used for different work. A discussion of improving system performance is beyond the scope of this book. However, there are several things you can do to maintain the performance level of your system:

- Keep directory files small.
- Reorganize file systems.
- Reconstruct minidisks.

This section covers each of these system maintenance tasks. (For a discussion of system configuration parameters that affect performance, see "Generating a New Kernel" on page 3-33.)

Keeping Directory Files Small

Directories that are larger than 10 ***logical*** blocks are very inefficient. A 10-block directory can contain over 1200 entries. (Each logical block can contain 2048 bytes of data, and each entry requires 16 bytes.) Thus, there usually is little reason to have larger directories. However, large directories (directories that occupy more than 10 blocks) can occur when, for example:

- A user has a large number of closely-related small files.
- A user does not understand how to use a directory structure to organize files.
- A directory (such as **/usr/news**) is shared by several users.
- A command or program writes its output to a new file (or files) each time it runs.

Your first tactic for keeping directory files small should be to inform all users that their directories should contain no more than a certain number of directories (a number well below the 1200 that could be possible). At the same time, you should make certain that all users understand how to use directory structures to organize files. (For information on how to use directory structures to organize files, see *Using the AIX Operating System.*)

Your second tactic should be to periodically search all file systems for directories larger than 10 blocks. If any large directories exist, the following **find** command displays a list of them:

```
find / -type d -size +10 -print
```

Simply removing files from a directory does not make that directory smaller; rather, it leaves the directory entries for the removed files vacant and ready to be assigned to other files.

To make a directory smaller, you must:

1. Make sure the directory has an acceptable number of entries. (For convenience, you might establish a 100-file directory size guideline.)

   If you need to remove files from the directory, you can use either the **rm** command (to remove files from the system) or the **mv** command (to move files to another directory).

2. Create a compacted directory or directory structure (one that does not contain the vacant entries).

   - To compact a single directory:

     a. Create a new directory with the **mkdir** command.

     b. Move (**mv**) or copy (**cp**) the files from the old directory into the new directory.

     c. Remove all files from the old directory (**rm**) and then remove the old directory (**rmdir**).

   - To compact a complete file system, use the **dcopy** command to copy the entire old file system to a new file system. For information about **dcopy**, see **dcopy** in *AIX Operating System Commands Reference.*

Reorganizing File Systems

When you create a file system, its free list is organized for maximum efficiency. However, as you create, delete, link, unlink, copy, and move files on the file system, the free list becomes less and less organized. At the same time, the physical location of data on the disk becomes less organized. The less organized the free list and data are, the longer average time it takes for the system to access files and directories in the file system. Reorganizing the free list and data of a file system often can improve system response time.

Reorganizing Only the Free List

Two flags cause the **fsck** command to reorganize free lists: **-s** and **-S**.

The **-s** flag causes **fsck** to reorganize the free list regardless of the condition of the file system. The **-S** flag causes **fsck** to reorganize the free list if **fsck** finds no inconsistencies in the file system.

If you only want to reorganize the free list, **fsck -S** is the quickest way to do so. You should run **fsck -S** regularly, perhaps once per week or twice per month, depending upon how heavily your system is used.

Note: Always run **fsck** on unmounted file system.

For more information about the **fsck** command, see "Checking and Repairing File Systems—The fsck Command" on page 3-6 and **fsck** in *AIX Operating System Commands Reference.*

Reorganizing Data and the Free List

There are two ways to reorganize both the data and the free list of a file system:

- **backup**, **mkfs**, and **restore**

  1. Back up the complete file system (with the **backup** command).
  2. Make a new file system on the minidisk that held the file system (with the **mkfs** command).
  3. Restore the backed up file system to the new file system (with the **restore** command).

  This procedure for reorganizing free lists is most appropriate when done in conjunction with routine level 0 system backups. For more information on **backup**, backup levels, and **restore**, see "Backing up Files and File Systems" on page 2-45. For more information on **mkfs**, see "Creating and Mounting File Systems" on page 2-39. Also see **backup**, **restore**, and **mkfs** in *AIX Operating System Commands Reference.*

- **dcopy**

  Besides compacting directories (as is discussed under "Keeping Directory Files Small" on page 4-29), the **dcopy** command also reorganizes a file system's free list. The main purpose of **dcopy** is to reorganize the file system for faster access times. It is also useful for copying an existing file system into a new file system (for example, when you move a file system from one minidisk to another, perhaps to balance the load among several fixed-disks). In such cases, the effect that **dcopy** has on directories and free lists is of secondary importance.

  However, if you want to use **dcopy** to reorganize the free list and compact directories, but do not want to move the file system, first use **dcopy** to copy the file system to an available minidisk, and then use either **dcopy** or **dd** to copy the file system back to its original minidisk. (For more information on the **dd** command, see **dd** in *AIX Operating System Commands Reference.*)

Handling the minidisk full Condition

Your system consists of a certain number of separate file systems, each of which resides on its own minidisk. The AIX Operating System consists of six minidisks: **/**, **/usr**, **/u**, **/tmp**, **/vrm**, page space, and dump. Additional minidisks may be created by the user.

The size of each minidisk is set when when the minidisk is created. A minidisk that is too small for its purpose can significantly reduce your system's performance. A minidisk can be too small either because it was not made large enough initially (the problem shows up immediately) or because it has become full with use (the problem may appear gradually).

If you suspect that one or more minidisks are becoming too full, enter the **df** command to determine, for each mounted file system:

- Total number of disk blocks
- Number of disk blocks free
- Number of disk blocks used.

Note: The **df** command (without arguments) only reports information about file systems with stanzas in **/etc/filesystems** that contain the line `df=true`.

In some cases, you may be able to correct a minidisk size problem by removing files from the file system. Often, however, you will need to reconstruct one or more minidisks, increasing their size as necessary. The remainder of this section describes how to recover from a `minidisk full` condition, whether it results from the installation of a program or from free space being used up over time.

Note: If you are not familiar with the **minidisks** command, please see the information on creating minidisks in *IBM RT Installing and Customizing the AIX Operating System* before you continue with this section.

To Recover from a minidisk full Condition

1. Use the **minidisks** command to determine the amount and location of existing free space.
2. Determine the amount of free space required.
3. Determine whether sufficient free space exists to meet your requirements.
4. Follow an appropriate recovery procedure.

If you are installing a licensed program, the documentation supplied with the program should state how many blocks the program requires and which file systems (minidisks) must contain those blocks. Thus, you know how much free space such a program requires. However, if a minidisk has simply filled up over time, it is not so easy to determine how much of the available free space you will allocate to it; you will have to make this decision based on your overall understanding of how your system is used.

If there is not sufficient free space to meet your requirements, you have two alternatives when faced with the minidisk full condition:

- Remove a sufficient number of existing files to create free space.
- Add additional storage to your system (that is, install another fixed disk).
- Replace an existing fixed disk with a larger one.

If there is sufficient free space available to meet your requirements, you can use one of the following four procedures to recover from a minidisk full condition:

- Expand a subdirectory into a file system.
- Create and mount a new licensed program file system.
- Expand a file system.
- Reconstruct existing minidisks.

Expanding a Subdirectory into a File System

This procedure involves creating a new minidisk, moving all data from an existing subdirectory into its own file system on the new minidisk, and then mounting the new minidisk on the original file system subdirectory.

Note: For this procedure to be successful, the subdirectory to be expanded must have no mounted subdirectories and there must be sufficient contiguous free space for a copy of the complete subdirectory plus the new data.

Following are the steps for expanding a subdirectory into a file system:

1. Use the **add** subcommand of **minidisks** to create a new minidisk large enough to hold the old subdirectory plus the new requirements. For the mount directory, specify the original subdirectory name.
2. Mount the new minidisk on a temporary directory (for example, **mount /dev/hd*n* /tmp/lpp**).
3. Copy the original subdirectory to the new file system. (To copy everything under the current subdirectory to the new file system, you can use the command: **find . -print | cpio -pdvm** *dirname*.)
4. Compare the original subdirectory to the new one (use the **dircmp** command).
5. Unmount the new file system from its temporary directory (for example, **umount /tmp/lpp**).
6. Use **fsck** to check the consistency of the new file system.
7. Delete files from the original subdirectory. (Enter **pwd** to verify that the original subdirectory is your current directory and then enter **rm -r *** to delete all files under that subdirectory.)
8. Use the **cd** command to move to the parent directory.
9. Mount the new file system onto the original subdirectory.

Creating and Mounting a New licensed program File System

This procedure involves creating a minidisk large enough for the new requirements, mounting that minidisk onto a new subdirectory of an existing file system, and installing the new files onto the new file system.

Note: For this procedure to be successful, there must be enough contiguous free space on the fixed disk to meet the new requirements.

Following are the steps for creating and mounting a new licensed program file system:

1. Use the **add** subcommand of **minidisks** to create a new minidisk large enough to meet the requirements of the new licensed program and specify the minidisk mount directory (for example, **/usr/lpp/***lppname*).
2. Create the minidisk mount directory (for example, **mkdir /usr/lpp/***lppname*).
3. Mount the new file system onto the minidisk mount directory (for example, **mount /usr/lpp/***lppname*).

Note: If you create a separate minidisk for each licensed program, you could eventually reach the maximum number of minidisks (57) and perhaps experience problems due to fragmentation of disk space. Therefore, it is good practice to merge minidisks periodically, if possible.

Expanding a File System

This procedure involves transferring all data from a mounted file system to a file system on a new, larger minidisk. There must be enough contiguous free space to hold the old file system and the new data. Following are the steps for expanding a file system:

1. Start the maintenance system from the AIX Operating System Installation/Maintenance diskette. (For information on starting and using the maintenance system, see "Running the Maintenance System" on page 2-5.)
2. Select `Use Maintenance Commands`.
3. Select `Backup commands`.
4. Select `Backup a file system` and back up the file system that you are expanding.

   Note: Do not use the `Backup a minidisk image` option.
5. Select `Delete a fixed disk minidisk` and write down the `Minidisk IODN` for the minidisk to be deleted—this information is required to complete the procedure. Delete the minidisk.

6. Select `Create a fixed disk minidisk`:
   a. Select `Specify IODN` and supply the IODN that you noted in the previous step.
   b. Select `No preference` to cause the minidisk to be created in the first disk area large enough to accommodate it. (Do not use the `beginning, middle, end` allocation suboptions.)
   c. Specify the number of blocks necessary to meet your new requirements.
7. Select `Make a file system` to create a file system on the new minidisk. Select the same IODN that the original file system had, the one you noted above.
8. Select `Restore commands`.
   a. Select `Restore a file system`
   b. Restore the old file system to the new minidisk, using the same IODN.
9. Select `Check a file system` and check the structure of the new file system.
10. Remove the AIX Operating System Installation/Maintenance diskette and start the system from the fixed disk.

Rearranging Existing Minidisks

With this procedure, you can physically rearrange the minidisks on the fixed disk(s). Use this procedure if:

- There is enough total free space to meet your requirements and
- You cannot use any of the previous procedures because the free space is not contiguous.

Following are the steps for rearranging existing minidisks:

1. Start the maintenance system from the AIX Operating System Installation/Maintenance diskette. (For information on starting and using the maintenance system, see "Running the Maintenance System" on page 2-5.)
2. Select `Use Maintenance Commands`.
3. Select `Backup commands`.
4. Select `Backup a file system` and back up the file systems that you are expanding. (If you want to arrange minidisks so that all of the free space is in one area, backup all minidisks between the first and last areas of free space on a particular disk.)

   Note: Use the `Backup a minidisk image` option only for minidisks that are not AIX file systems.

5. Select `Delete a fixed disk minidisk` and write down the `Minidisk IODN` and number of blocks for each minidisk to be deleted—this information is required to complete the procedure. Delete the minidisk.

6. Select `Create a fixed disk minidisk` to create a new minidisk to replace each of the minidisks that you deleted:

   a. Select `Specify IODN` and supply the IODN that you noted in the previous step.

   b. Select `No preference` to cause the minidisk to be created in the first disk area large enough to accommodate it. (Do not use the `beginning`, `middle`, `end` allocation suboptions.)

   c. Specify the number of blocks that you noted in the previous step.

   d. Repeat these steps for each of the new minidisks.

7. Select `Make a file system` and create a file system on each new minidisk that requires one (that is, the ones that originally had file systems). Use the same IODN that the original file system had, the one you noted above.

8. Select `Restore commands`.

   a. Select `Restore a file system`

   b. Restore the old file system to the new minidisk, using the same IODN.

   c. Repeat these steps for each of the new file systems.

   Note: To restore minidisks that do not have AIX file systems (the ones that were backed up with the `Backup a minidisk image` option), select the `Restore a minidisk image` option.

9. For all minidisks that contain AIX file systems, select `Check a file system` and check the structure of the new file system.

10. Remove the AIX Operating System Installation/Maintenance diskette and start the system from the fixed disk.

Logging in Automatically

Generally, it is best to use the standard login and password system to protect your system from unauthorized access. However, if your situation does not require this security, you can modify your system so that it logs you in automatically each time you start it. To make your system log you in automatically, create a file named `/etc/autolog` that contains a valid user name (that is, a user name that is in the password file). For information about users and the password file, see "Managing User Accounts" on page 2-13.

Notes:

1. You can use the automatic log in only for the main display stations (the console).
2. The automatic log in is disabled if you enable the controlled access mode.

Using the IBM 6156 Portable Disk Drive

This section describes how to use the IBM 6156 Portable Disk Drive (PDD). The PDD consists of module ***enclosures*** connected to the RT, and removable disk ***modules***; it provides portable, high-density disk storage.

This section includes details about:

- The hardware that constitutes the PDD.
- The process of ***inserting*** modules into an enclosure and ***configuring*** them into the AIX Operating System.
- The process of ***removing*** modules from an enclosure and ***unconfiguring*** them from the AIX Operating System.
- Manual procedures to help you keep track of what data is stored on each module.
- The use of PDD modules in the ***Initial Program Load (IPL)*** process.

For most types of disk drives, configuring and unconfiguring are done infrequently, and usually by the system administrator. The PDD, however, is designed for frequent and quick module insertion and removal. This section describes these processes at the user level.

Introduction to the IBM 6156 Portable Disk Drive

The IBM 6156 Portable Disk Drive (PDD) functions much the same as other internal and external hard-disk drives available for the RT. However, the PDD provides the advantage of removable, portable disk modules, which you can move, store, ship, secure, or transfer to other RT systems. The PDD is intended for users who:

- Store very large amounts of data, but only need to use parts of it at a time
- Routinely share large amounts of data with users of other RT systems
- Must remove and secure large data files when they are not in use.

The Portable Disk Drive consists of two components: an enclosure and a module. Your RT system can have only one enclosure, but it can use as many modules as you need.

PDD Enclosure A PDD enclosure contains power supplies, interfacing circuitry, and mechanical and electrical connectors for one or more PDD modules. An enclosure is connected by a cable to the RT system unit. PDD enclosures are available in two models, which differ only in the number of ***module locations*** they contain. Model 001 has one module location; model 003 has three. Each RT system can have one PDD enclosure, of either model.

PDD Module A PDD module contains a disk drive mechanism, storage media, and mechanical and electrical connectors to a PDD enclosure.

Configuring a Module

Configuring a PDD module is the process of telling AIX that the module is available, and where to find it. Most disk drives are configured at the time of installation, and remain configured for a long period of time. However, PDD modules are designed for frequent and quick insertion and removal, so they are configured and unconfigured often. You must configure a PDD module before you use it, and unconfigure the module before you remove it from its enclosure. The commands and procedures for configuring PDD modules are similar to those for configuring other types of disk drives.

Configuring a Module

1. Insert the module into any available module location in the enclosure. Be sure the power switch on the enclosure is turned on.
2. Use the **varyon** command to configure the module:

   `varyon -d hdisk`*num*

   where *num* is the drive (module location) number.
3. If desired, use the **minidisks** command to change mount directories of minidisks.

Inserting the Module

Before you can configure a PDD module, you must physically insert it into any available module location in the enclosure. Be sure the power switch on the enclosure is turned on. If you need help using the enclosure, see *IBM 6156 Portable Disk Drive Setup, Operations, and Technical Information*, which was supplied with the enclosure.

Each module location in the enclosure has its own drive name. Drive names have the form **hdisk***num*, where *num* is an integer from 0 through 5. (A model 001 enclosure uses one drive name; a model 003 enclosure uses three drive names.) To use the **varyon** command, you must know the drive name of the module location into which you inserted the module. If the module location is not labeled with its drive name, you can find out the name(s) assigned to the enclosure by referring to *IBM 6156 Portable Disk Drive Setup, Operations, and Technical Information*, which was supplied with the enclosure.

The varyon Command

Note: You can use the **varyon** command to configure an IBM 9332 Direct Access Storage Device as well as a PDD module.

To configure a PDD module, use the **varyon** command. **varyon** renames minidisks on the module (if you request it), updates AIX tables to show that the module and its minidisks are available, and where to find them, updates the AIX configuration files to show current configuration parameters, and performs a mount operation on the minidisks[3], which connects them into the AIX file system.

Note: Regardless of the actions you select, **varyon** does not change the data on a PDD module in any way. The only thing **varyon** ever changes on a module is the names or IODNs of its minidisks.

---

[3] **varyon** mounts the minidisks on a module if the module's stanza in file **/etc/filesystems** includes the attribute **vmount = true**.

Enter the **varyon** command as follows:

```
varyon -d hdisknum
```

where *num* is the drive (module location) number; it must be an integer from 0 through 5. **varyon** proceeds as described in "Inspecting for the Correct Module."

Inspecting for the Correct Module

After you enter the **varyon** command, **varyon** displays the module's minidisk configuration, as recorded on the module (unless you specify the **-q** command flag), as shown in Figure 4-6.

```
MD    MD     Blk   Number  MD
Name  IODN   Size  Blocks  Type
hd13  20010  512   1000    AIX
hd14  20011  512   1000    AIX
hd15  20012  512   2000    AIX
hd16  20013  512   500     CPPRC
hd17  20014  512   1000    AIX
hd18  20015  512   10000   AIX
hd19  20017  512   1000    AIX

To EXIT the varyon command, Press F3.
To CONTINUE, press Enter.
```

Figure 4-6. varyon Command – Listing of Disk Configuration

This listing shows, for all minidisks on the module that **varyon** recognizes[4], the name, IODN (I/O Device Number), block size, number of blocks, and type for each minidisk.

Examine this listing, and see if it appears to describe the data you are expecting. If any of the names or parameters do not appear to match the data you are expecting, you may have inserted the wrong module. (For a detailed discussion of mismatched configuration information, see "Configuration Conflicts" on page 4-52.)

4 **varyon** recognizes minidisks whose IODNs are in the range 20001 – 20064, inclusive. This is the same range that is processed by the **minidisks** command. Minidisks that **varyon** does not recognize should be configured by the Initial Program Load (IPL) process. For more information, see "Using Modules at IPL Time" on page 4-55.

After you examine the listing of the disk configuration (see Figure 4-6 on page 4-40), you can do two things:

1. If you think you have inserted the wrong module, press **F3** to terminate **varyon**. Remove the module, insert the correct one, and enter the **varyon** command again.
2. If you think you have inserted the correct module, press **Enter** to proceed. **varyon** does one of two things:
   a. If the AIX configuration files contain no information about any minidisks on the module, **varyon** proceeds without prompting you further. **varyon** updates AIX tables to show that the module and its minidisks are available, and where to find them, and updates the AIX configuration files. **varyon** then ends.
   b. If the AIX configuration files already contain information about any minidisk on the module, **varyon** proceeds as described in "Actions When Minidisks Are Already Defined."

Actions When Minidisks Are Already Defined

If you told **varyon** to configure the module, and the AIX configuration files contain information about any minidisk on the module (see step 2b in "Inspecting for the Correct Module"), **varyon** displays a second listing. Figure 4-7 shows an example.

```
The following minidisks defined on your disk have the same NAME or IODN as
minidisks defined in your system configuration.

Disk Configuration                           System Configuration
MD    MD     Blk   Number  MD                MD    MD     Blk   Number  Mount
Name  IODN   Size  Blocks  Type              Name  IODN   Size  Blocks  Directory
hd13  20010  512   1000    AIX               hd13  20010  512   1000    /proj115
hd14  20011  512   1000    AIX               hd14  20011  512   1000    /proj18
hd15  20012  512   2000    AIX               hd15  20012  512   2000    /acctg107
hd16  20013  512   500     AIX               hd16  20013  512   500     /tmp/directory/hd16
hd17  20014  512   1000    AIX               hd17  20014  512   1000    /tmp/directory/hd17
hd18  20015  512   10000   AIX               hd18  20015  512   10000   /tmp/directory/hd18
hd19  20017  512   1000    AIX               hd19  20017  512   1000    /tmp/directory/hd19

Are the minidisks on your disk the same minidisks defined in your system?
To EXIT the varyon command, Press F3.
To CONTINUE, type yes or no and press Enter.
```

Figure 4-7. varyon Command – Listing of Disk Configuration vs. System Configuration

This listing shows information **only** for minidisks whose name or IODN are already defined in the AIX configuration files. The listing shows information from two sources: from the module itself, and from corresponding entries in the AIX configuration files. This permits you to see whether the information on the module matches the information in the AIX configuration files. Look for two things:

- Check the `MD Name`, `MD IODN`, `Blk Size`, and `Number Blocks` parameters for each minidisk. If any parameter in the `Disk Configuration` section does not match the same parameter in the `System Configuration` section, for any minidisk in this listing, **varyon** cannot configure the module using the parameters shown. If this occurs, go to step 1 in this section, or step 3 in this section.
- If all the parameters **do** match, examine the `Mount Directory` column, and see if the names appear to describe the data you are expecting. If any of the names or configuration parameters do not appear to match the data you are expecting, you may have inserted the wrong module. If this occurs, go to step 1 in this section, or step 2 in this section.

For more information about configuration conflicts, see "Configuration Conflicts" on page 4-52.

After you examine the listing of the disk configuration vs. the system configuration (see Figure 4-7 on page 4-41), you can do three things:

1. If there is an obvious conflict of configuration parameters, or if you want to terminate **varyon** for any other reason, press **F3**. **varyon** terminates, and makes no changes to the module, to AIX tables, or to the AIX configuration files.
2. If you want to configure the module using the parameters shown, enter `yes`. **varyon** proceeds as described in "Configuring the Module Using the Current Configuration" on page 4-43.
3. If there are conflicts between parameters on the module and in the AIX configuration files, and you do not want to change the existing information in the AIX configuration files, **varyon** can rename the minidisks on the module and configure the module with the new names. To do so, enter `no`. **varyon** proceeds as described in "Configuring the Module Using a New Configuration" on page 4-45.

Configuring the Module Using the Current Configuration

If you told **varyon** to configure the module using its current configuration parameters (in step 2 in "Actions When Minidisks Are Already Defined"), **varyon** does one of three things:

1. If all configuration parameters for all minidisks are the same, both on the module and in the AIX configuration files, and if none of the minidisks named are already configured, **varyon** configures the module as shown. **varyon** updates AIX tables to show that the module and its minidisks are available, and where to find them, and updates the AIX configuration files. **varyon** then ends.

2. If any of the minidisks shown in the listing of disk configuration vs. system configuration (see Figure 4-7 on page 4-41) is already configured, **varyon** will not configure the module. **varyon** issues an error message and terminates. You can resolve this problem by doing one of the following:

   - Execute **varyon** again, and ask it to rename the minidisks on the module (see step 3 in "Actions When Minidisks Are Already Defined").

   - Find out what module(s) contain the minidisk(s) that conflict with this module, and unconfigure those module(s) (see "Unconfiguring a Module" on page 4-48).

   - Find out what module(s) contain the minidisk(s) that conflict with this module, and delete the conflicting minidisks from those modules. Use the **minidisks** command.

3. If none of the minidisks named is already configured, but some configuration parameters do not match between the module and the AIX configuration files, for any minidisk, **varyon** displays a listing showing the conflicts. Figure 4-8 shows an example.

```
The varyoff command cannot complete because the following minidisk information
  found on your disk conflicts with the information found in your system
  configuration.  View the file /etc/varyon.out for more detail.

Disk Configuration                 System Configuration
MD    MD     Blk   Number  MD      MD    MD     Blk   Number  Mount
Name  IODN   Size  Blocks  Type    Name  IODN   Size  Blocks  Directory
hd33  20101  512   1000    AIX     hd33  20101  512   10000   /u/smith
hd34  20102  1024  1000    AIX     hd34  20102  512   1000    /u/jones
hd35  20103  512   1000    AIX     hd46  20103  512   1000    /u/roberts
hd36  20104  512   1000    AIX     hd36  20228  512   1000    /u/jackson

To EXIT the varyoff command, Press Enter.
```

Figure 4-8. varyon Command – Conflict Between Disk Configuration and System Configuration

In this example, four conflicts have occurred:

For hd33, the `Number Blocks` parameters do not match.
For hd34, the `Blk Size` parameters do not match.
For hd35, the `MD Name` parameters do not match.
For hd36, the `MD IODN` parameters do not match.

Conflicts of this kind usually indicate that you have inserted the wrong module. Do one of the following:

- If you think you have inserted the wrong module, terminate **varyon**. Remove this module, find the correct module, insert it, and enter the **varyon** command again.
- If you think you have inserted the correct module, terminate **varyon**. Enter the **varyon** command again, and use it to rename the minidisks on the module (see step 3 in "Actions When Minidisks Are Already Defined").

The only action **varyon** allows at this point is to press **Enter**. When you are finished looking at the listing of conflicts between disk configuration and system configuration, press **Enter**, and **varyon** terminates.

Configuring the Module Using a New Configuration

If you told **varyon** to rename the minidisks on the module and configure the module with the new names (in step 3 in "Actions When Minidisks Are Already Defined"), **varyon** proceeds as described in this section.

Note: **varyon** does not do the renaming immediately. It first displays a listing of the current configuration and the proposed new configuration, then asks you to confirm the changes.

varyon assigns new minidisk names by searching the file **/etc/system** for names that are not yet assigned. **varyon** starts with the lowest name, **hd7** (names **hd0** through **hd6** are reserved for AIX Operating System minidisks), and searches toward the highest name, **hd63**.

After completing its search, **varyon** does one of two things:

1. If **varyon** cannot find enough unassigned names to rename all the minidisks on this module, it issues an error message and stops. You can resolve this problem by doing one of the following:
   - Unconfigure one or more other modules, removing their configuration information from the AIX configuration files (by using the **varyoff** command with the **-r** command flag), until there are enough unassigned names to allow **varyon** to configure this module. See "Unconfiguring a Module" on page 4-48.
   - Delete minidisks from other modules until there are enough unassigned names to allow **varyon** to configure this module. Use the **minidisks** command.
2. If **varyon** does find enough unassigned names to rename all the minidisks, it displays a listing showing the current configuration on the module, and the proposed new configuration. Figure 4-9 shows an example.

```
To fit your system configuration, the following changes will be made to the
  minidisk configuration information contained on your disk.

Current Disk Configuration           Modified Disk Configuration
MD    MD     Blk   Number  MD        MD    MD     Blk   Number  MD
Name  IODN   Size  Blocks  Type      Name  IODN   Size  Blocks  Type
hd13  20010  512   1000    AIX       hd8   20010  512   1000    AIX
hd14  20011  512   1000    AIX       hd23  20011  512   1000    AIX
hd15  20012  512   2000    AIX       hd24  20012  512   2000    AIX
hd16  20013  512   500     AIX       hd25  20013  512   500     AIX
hd17  20014  512   1000    AIX       hd26  20014  512   1000    AIX
hd18  20015  512   10000   AIX       hd42  20015  512   10000   AIX
hd19  20017  512   1000    AIX       hd43  20017  512   1000    AIX

To EXIT the varyon command, Press F3.
To CONFIRM the disk changes, press Enter.
```

Figure 4-9. varyon Command – Listing of Current Disk Configuration vs. Modified Disk Configuration

You do not need to check this listing for parameter conflicts; all parameter values are the same except `MD Name` or `MD IODN`. (The listing does not show the new mount directories that **varyon** will assign to the minidisks. **varyon** will assign default mount directories of the form **/tmp/directory/hd***num*, where *num* is the new minidisk number.)

After you examine the listing of the current disk configuration vs. the modified disk configuration (see Figure 4-9), you can do two things:

a. If you do not want to make the changes shown, press **F3**. **varyon** terminates, and makes no changes to the module, to AIX tables, or to the AIX configuration files.

b. If you want to make the changes shown, enter `yes`. **varyon** records the new names and IODNs on the module, updates AIX tables to show that the module and its minidisks are available, and where to find them, and updates the AIX configuration files. **varyon** then ends.

   After **varyon** ends, you may want to change the default mount directories it assigned, by using the **minidisks** command (see "Changing Mount Directories of Minidisks" on page 4-47).

varyon Command Flags

varyon has two command flags that control how it operates. Figure 4-10 describes the flags and what they do.

Flag	Action	Example
-q	Configures the module exactly as it is defined in the configuration files. Does not prompt for further information. Functions exactly as if you answered `yes` in step 2 on page 4-42 in "Actions When Minidisks Are Already Defined."	`varyon -d hdisk3 -q`
-c	Suppresses file consistency check.	`varyon -c -d hdisk2 hdisk3`

Figure 4-10. varyon Command Flags

New Modules

After you configure a new module, you must create at least one minidisk on it. New modules contain no minidisks, and you cannot use the module until it contains at least one minidisk. For information on creating minidisks, see "Changing Mount Directories of Minidisks."

Changing Mount Directories of Minidisks

After you configure a module with **varyon**, you may want to change the mount directories assigned to some of your minidisks. This is particularly true if you have minidisks with default mount directories of the form **/tmp/directory/hd***num*. Such names are not descriptive of the data on a minidisk, and you may want to assign one that **is** descriptive.

To change the mount directory for a minidisk, use the **minidisks** command. You can use **minidisks** for other functions as well, such as creating and deleting minidisks and changing minidisk configuration parameters. If you just configured a new PDD module, which contains no minidisks, you must create at least one minidisk on the module before you can use it. The **minidisks** command is described in *Installing and Customizing the AIX Operating System.*

Unconfiguring a Module

Unconfiguring a PDD module is the process of telling AIX that the module is no longer available for use. You must unconfigure a PDD module before you remove it from its module location.

Unconfiguring a Module

1. Use the **varyoff** command to unconfigure the module:

   ```
   varyoff -d hdisknum
   ```

   where *num* is the drive (module location) number.

2. Remove the module from its module location.

The varyoff Command

Note: You can use the **varyoff** command to unconfigure an IBM 9332 Direct Access Storage Device as well as a PDD module.

To unconfigure a PDD module, use the **varyoff** command. **varyoff** does the opposite function of **varyon**. **varyoff** performs an unmount operation on the minidisks, which disconnects them from the file system, then modifies AIX tables to show that the module and its minidisks are no longer available.

If you request it, **varyoff** can also remove all information about the module from the AIX configuration files. If you intend to use this module again in the near future, leave its information in the configuration files. That way, the **varyon** command can configure the module and mount all its minidisks with the same mount directories they had before. However, you may intend to put the module into storage for a long period of time, or transfer it to a user of another RT system. If so, you may prefer to remove its information from the AIX configuration files. To do so, use the **-r** command flag, described in "varyoff Command Flags" on page 4-49. **varyoff**'s default action is to keep minidisk information in the configuration files.

To unconfigure a module and keep information about it in the AIX configuration files, enter **varyoff** as follows:

```
varyoff -d hdisknum
```

where *num* is the drive (module location) number; it must be an integer from 0 through 5.

varyoff Command Flags

varyoff has two command flags that control how it operates. Figure 4-11 describes the flags and what they do.

Flag	Action	Example
-r	Removes configuration information from the files **/etc/system** and **/etc/filesystems**.	`varyon -d hdisk3 -r`
-c	Suppresses file consistency check.	`varyon -c -d hdisk2 hdisk3`

Figure 4-11. varyoff Command Flags

Removing the Module

After you unconfigure a PDD module, remove it from the module location in the enclosure. If you need help removing the module, see *IBM 6156 Portable Disk Drive Setup, Operations, and Technical Information*, which was supplied with the enclosure.

Managing Modules

Managing PDD modules is easy when you have only a few minidisks. If you have 64 or fewer minidisks among all your hard disks (including all your PDD modules), and you do not exchange modules with users of other RT systems, you may not have to do any managing at all. All the information you need is stored in the AIX configuration files, and **varyon** configures your minidisks and mounts them to the correct mount directories every time. Once your configuration is stable, you may want to use the **-q** command flag with **varyon** to suppress the normal prompting.

However, if you exchange PDD modules with users of other RT systems, you may have to use additional procedures to manage modules. If you have more than 64 minidisks among all your hard drives and PDD modules (including those you exchange with users of other RT systems), you **will** have to use additional procedures.

Manual Procedures

Keeping track of the data on PDD modules will be much easier if you follow the manual procedures recommended here. If you have only a few minidisks, and do not exchange modules with users of other RT systems, these procedures are helpful, but not important. However, if you have many modules and minidisks, these procedures can help prevent some confusion. And if you exchange modules with users of other RT systems, they are almost essential.

The following procedures are recommended:

- Label each module. Labels are supplied with the modules; the best place to put them is on the front of the modules. Each label should clearly describe the data on the module: it should include information such as names of minidisks, their assigned mount directories, and other configuration parameters. You may also want to include identifying information such as the owner's name, telephone number, address, and electronic mail code.
- Whenever you create or modify the minidisk configuration of a PDD module, use the **minidisks** command to print a list of the configuration parameters. Changes include those made with the **minidisks** command, and renaming done by the **varyon** command. Attach the list to the PDD module.
- If the label and the list of configuration parameters do not include enough information to describe the data on the module, write down whatever extra information is needed. Attach it to the module.

AIX Configuration Files

The AIX Operating System uses two files to keep track of disks and minidisks that are part of the current system configuration:

/etc/filesystems Contains one ***stanza*** (entry) for each file system name, each of which resides on a minidisk or diskette. For minidisks, each stanza begins with its file system name, and lists its minidisk name and other parameters.

For example, a stanza could be:

```
/proj117:
    dev        = /dev/hd7
    size       = 512
    mount      = false
    vmount     = true
    check      = false
    vcheck     = true
    .
    .
```

/etc/system Contains one stanza for each minidisk. Each stanza lists the name of the drive (module location) into which the module was inserted the last time it was configured, the module's IODN (I/O Device Number), and other parameters. For example, a stanza could be:

```
hd7:
         fd = hdisk4
         iodn = 20034
         noipl = true
         keep = true
         driver = disk
         .
         .
```

When you configure a PDD module, **varyon** checks these two files to see if stanzas already exist for each minidisk on the module. If they do exist, **varyon** uses them as described in "Configuring a Module" on page 4-38. If either stanza does not exist, **varyon** creates it—one stanza for each minidisk in each of the two files. When you unconfigure a module, **varyoff** normally leaves these stanzas in the files. **varyon** deletes them only if you specify the **-r** command flag for **varyoff**. Thus, stanzas in the configuration files do not show whether a minidisk is currently configured. They show only that the minidisk was configured at some time in the past, and what its parameters were at that time. The list of disks and minidisks that are currently configured is stored in internal AIX Operating System tables.

Note: When you install a new version of the AIX Operating System, the installation process replaces the files **/etc/system** and **/etc/filesystems**. After you finish installing the new version, use **varyon** to configure each module for which you want information in the AIX configuration files.

For more information about the file **/etc/filesystems**, see "Information about File Systems—The /etc/filesystems File" on page 2-36 or *AIX Operating System Technical Reference*. For more information about the file **/etc/system**, see *AIX Operating System Technical Reference*.

Minidisk Names

Minidisks are partitioned areas on a hard disk. When you create a minidisk, the **minidisks** command assigns it a name of the form **hd***num*, where *num* is an integer from 7 through 63. (Minidisk names **hd0** through **hd6** are reserved for AIX Operating System use. They normally reside on an internal hard disk, but they can reside on a PDD module. For more information, see "Using Modules at IPL Time" on page 4-55.) **minidisks** assigns names automatically, searching the file **/etc/system** for the first currently unassigned name. You cannot control the assignment of minidisk names.

Within your set of fixed hard disks and PDD modules, duplicate minidisk names can occur. The two most common reasons are:

1. The assignment of names depends on the current contents of **/etc/system**. If you unconfigure a module and use the **-r** command flag for **varyoff**, **/etc/system** no longer has entries for those minidisk names. **varyon** may then assign the same names to minidisks on other modules.
2. Modules created on other RT systems use the same set of names **hd***num*. If you receive modules that were created on other RT systems, they may include duplicates of names assigned by your system.

Duplicate minidisk names or IODNs are not a problem, until you try to configure a minidisk whose name or IODN is already present in the AIX configuration files. When this occurs, **varyon** cannot configure the module until you take some action to resolve the conflict.

Configuration Conflicts

A configuration conflict occurs when **varyon** cannot configure a module according to its configuration parameters on the module itself and in the AIX configuration files. There are three types of conflicts. If any of these conflicts occurs, you **must** resolve it in order to configure the module. **varyon** cannot proceed until you do so.

1. **varyon** will not configure a module if it contains a minidisk whose name or IODN is the same as the name or IODN of another minidisk that is currently configured. To resolve this problem, see step 2 on page 4-43 in "Configuring the Module Using the Current Configuration."
2. **varyon** will not configure a module if any configuration parameter of any of its minidisks is different from the same parameter for the same minidisk in the AIX configuration files. To resolve this problem, see step 3 on page 4-43 in "Configuring the Module Using the Current Configuration."
3. **varyon** will not configure a module if you try to rename its minidisks, and **varyon** cannot find enough unassigned names to rename all the minidisks. To resolve this problem, see step 1 on page 4-45 in "Configuring the Module Using a New Configuration."

Even if none of these conflicts occurs, it is still possible that you have inserted a module other than the one you intended. When **varyon** displays the listing of the disk configuration vs. the system configuration (see Figure 4-7 on page 4-41), examine the mount directories shown in the `Mount Directory` column. If any of them do not seem to match the data that should be on this module, this module may contain minidisks with the same name and configuration parameters as another minidisk on another module. If you think this is the case, terminate **varyon**, remove the module, insert the module you think is the right one, and execute **varyon** again.

Conflicts usually occur when you have more than 64 minidisks among all disks on your RT system (including all your PDD modules), or when you are using a module that was created on another RT system. If you have more than 64 minidisks, or you are using modules created on other RT systems, you can avoid most conflicts by following the module management system described in "A Suggested Management System."

A Suggested Management System

You can avoid most module configuration conflicts by using the module management system described here. To establish the management system, follow these steps:

1. Figure out how many minidisk names **hd***num* are available on your RT system for use by PDD modules. This number will be 57 (**hd07** through **hd63**) minus the number of minidisks defined on other disk drives on your RT system. For example, if your RT system has the following minidisks defined:

   2 minidisks on internal drive hdisk0
   7 minidisks on internal drive hdisk1
   5 minidisks on external drive hdisk2

   then you have (57-2-7-5) = 43 minidisk names available for use by PDD modules.

2. Classify your PDD modules into two groups, ***frequent-use*** modules and ***transient-use*** modules:

   - Frequent-use modules are those that you use often, or for long periods of time. You will want to leave their configuration information in the AIX configuration files.
   - Transient-use modules are those that you use less often, or for shorter periods of time. You will want to delete their configuration information from the AIX configuration files whenever you unconfigure the module.

3. Divide the number of available minidisk names between the frequent-use and transient-use groups. To do so, follow these steps:

   a. For the frequent-use group, add up the number of minidisks on the modules in that group.

   b. For the transient-use group, determine the largest number of minidisks you would expect to have configured at any one time.

c. If the total of the numbers from steps 3a and 3b is less than or equal to the number of available names (step 1), go on to step 4 on page 4-54.

d. If the total of these two numbers is more than the number of names that are available, adjust the sizes of one or both groups.

Example: You have 43 minidisk names available for use by PDD modules. You have classified four of your modules as frequent-use, and they contain a total of 37 minidisks. You also have other modules that you have classified as transient-use; from this group, you expect to use at most two modules at a time, which will contain a total of no more than 14 minidisks. However, the sum of those two group totals is (37+14) = 51, which exceeds the 43 names that you have available. After some thought, you decide that you can reclassify one module, containing 5 minidisks, from the frequent-use group to the transient-use group. This leaves (37-5) = 32 frequent-use minidisks. The sum of the two group totals is now (32+14) = 46, which is still more than 43. You then decide that you can get by with only 11 transient-use minidisks at a time, bringing the sum of the group totals to exactly (32+11) = 43.

4. Label all the frequent-use modules. On the label on the front of each module, write "frequent use."

5. If the AIX configuration files on your RT system contain information about any PDD modules, delete that information. To do so, use the **varyon** command to configure each module, then unconfigure it, using the **varyoff** command with the **-r** command flag. When you finish, the AIX configuration files should contain no information about any PDD modules.

6. Add configuration information to the AIX configuration files for each frequent-use module. To do so, use the **varyon** command to configure the module, then unconfigure it, using the **varyoff** command (**do not** use the **-r** command flag). As you configure the modules, their minidisks' names or IODNs may conflict with other modules that you just configured, or with minidisks on other internal or external disk drives. If this occurs, tell **varyon** to rename the minidisks on the module that you are configuring.

When you complete these steps, the module management system is established. Your frequent-use modules are labeled, and the AIX configuration files contain their configuration parameters. If you use this management system, you normally will have configuration conflicts only if you try to configure too many transient-use minidisks at one time. If you need to configure more transient-use minidisks than you originally reserved space for, temporarily remove the configuration information for one or more of the frequent-use modules. To do so, configure the module(s) with **varyon**, then unconfigure it/them with **varyoff**, using the **-r** command flag. After you finish using the transient-use modules and unconfigure them, reconfigure the frequent-use module(s).

To successfully manage modules under this system, use the **varyon** and **varyoff** commands as described in Figure 4-12.

Module type	To configure	To unconfigure
Frequent-use	`varyon hdisk`*num* `-q` Configures the module using its current configuration; does not prompt you unless an error occurs.	`varyoff hdisk`*num* Leaves configuration parameters in the AIX configuration files.
Transient-use	`varyon hdisk`*num* Displays information and prompts you for the actions you want to take. If any minidisks are already defined in the AIX configuration files, tell **varyon** to rename all the minidisks. If you do so, use the **minidisks** command to print a list of the new configuration parameters. Use **minidisks** to assign new mount directories if desired.	`varyoff hdisk`*num* `-r` Removes configuration parameters from the AIX configuration files.

Figure 4-12. Configuring / Unconfiguring With Suggested Management System

This management system may not work well with your particular way of using modules. If so, another way to avoid configuration conflicts is to **always** remove configuration parameters from the AIX configuration files. If you always use the **-r** command flag with **varyoff**, the configuration files should not contain any old information that can cause a conflict. The advantage of this method is that you normally will have no configuration conflicts (unless you try to configure more than 64 minidisks at one time). The disadvantage is that, if you want the minidisks on a module to have descriptive mount directory names, you must use the **minidisks** command to assign those names every time you configure the module.

Using Modules at IPL Time

You can use PDD modules to store minidisks that are to be configured by the Initial Program Load (IPL) process, rather than by the **varyon** command. This section describes how to set up minidisks and modules for configuration by the IPL process, and how to configure them.

This section uses the following terms:

IPL-only minidisk
: A minidisk that is configured by the IPL process. Usually, a minidisk that is used by the AIX Operating System.

varyon-only minidisk
: A minidisk that is configured by the **varyon** command.

IPL-only module
: A PDD module containing only IPL-only minidisks.

varyon-only module
: A PDD module containing only varyon-only minidisks.

mixed-use module
: A PDD module containing a mixture of IPL-only and varyon-only minidisks. Mixed-use modules are considered undesirable, and should be corrected as described below.

You should keep IPL-only minidisks completely separate from varyon-only minidisks. If you mix IPL-only and varyon-only minidisks on the same module, you may be unable to access some of the minidisks. If you configure such a mixed-use module during IPL, the IPL process will not configure the varyon-only minidisks, and you cannot use the **varyon** command on a module that was configured during IPL; thus, the varyon-only minidisks are inaccessible. If you configure a mixed-use module with the **varyon** command, **varyon** will not process the IPL-only minidisks, and they are inaccessible.

Note: the distinction between IPL-only and varyon-only minidisks is **not** the same as the distinction between system and user minidisks. System minidisks should reside on an IPL-only module. However, user minidisks can reside on either a varyon-only module or an IPL-only module. Thus, you can use all the storage available on an IPL-only module: put system minidisks on it first, then put on as many user minidisks as you wish.

If you have any mixed-use modules, correct them by following these steps:

1. Get a list of the minidisks on the module. To do so, begin the process of configuring the module with the **varyon** command. When **varyon** displays the listing of the disk configuration (see Figure 4-6 on page 4-40), write down the names of the minidisks. Use this list as a check list, to be sure you set up all minidisks on the module correctly.
2. Decide whether each module will be a varyon-only module or an IPL-only module.
3. If the module will be an IPL-only module, go to step 5 in "Changing a Varyon-Only Module to IPL-Only" on page 4-57, and continue to the end of that procedure. If the module will be a varyon-only module, go to step 4 in "Changing an IPL-Only Module to Varyon-Only" on page 4-59, and continue to the end of that procedure.

Changing a Varyon-Only Module to IPL-Only

Use this procedure **only** for modules that are currently set up as varyon-only. If the module contains a mixture of varyon-only and IPL-only minidisks. follow the procedure described in "Using Modules at IPL Time" on page 4-55.

To change a varyon-only module to an IPL-only module, follow these steps:

1. Configure the module with the **varyon** command.
2. Use the **minidisks** command to print a list of all the minidisks on the module. Use this list as a check list, to be sure you set up all minidisks on the module.
3. Make backup copies of the files **/etc/system** and **/etc/filesystems**. You may need to restore these files from the backup copies if you make mistakes during the rest of the procedure.
4. Unconfigure the module with the **varyoff** command. **Do not** use the **-r** command flag; the configuration information needs to stay in the files.
5. Edit the file **/etc/system**. To do so, you must have superuser authority.
6. In the file **/etc/system**, edit the stanza for each minidisk on the module. Set the `noipl` attribute to the value `false`, as in this example:

   ```
   noipl = false
   ```

   This setting tells the IPL process that this minidisk **is** to be included.
7. If you want to change the mount or file consistency check attributes for any minidisks, edit the file **/etc/filesystems**. (To do so, you must have superuser authority.) Edit the stanza for the mount directory corresponding to each minidisk on the module. Make the following changes:

   a. If you want the IPL process to automatically mount the minidisk, set the following attribute values:

      ```
      mount = true
      ```

   b. If you want the IPL process to automatically perform a file consistency check on the minidisk, set the following attribute values:

      ```
      check = true
      ```

To configure the newly-established IPL-only module, follow these steps:

1. Shut down the RT system, using the **shutdown** command.
2. Insert the module into a module location. Be sure the power switch on the enclosure is turned on.
3. Start the IPL process by pressing the **Ctrl**, **Alt**, and **Pause** keys at the same time.

If you made any mistakes during this procedure, and the module is not configured as you wanted, go back to step 5 on page 4-57 and correct the problems.

From now on, configure this module using the procedure described in "Using an IPL-Only Module."

Using an IPL-Only Module

Once you set up a module as IPL-only, configure it **only** during the IPL process. Do not use the **varyon** or **varyoff** commands with an IPL-only module. To configure an IPL-only module, follow these steps:

1. Insert the module into a module location. You can insert the module into any module location in the enclosure, and you can put it in a different module location every time.
2. Insert the module into a module location **before** you begin the IPL process. Be sure the power switch on the enclosure is turned on.

   Note: you must insert **all** modules that you have set up as IPL-only, and turn on power to the enclosure, before you start the IPL process. If you omit any IPL-only modules, and they contain data that the IPL process needs, the IPL process may fail.
3. If AIX is currently running, shut it down with the **shutdown** command.
4. Begin the IPL process. To do so, either turn on the power to the RT system unit, or press the **Ctrl**, **Alt**, and **Pause** keys at the same time.

Changing an IPL-Only Module to Varyon-Only

Use this procedure **only** for modules that are currently set up as IPL-only. If the module contains a mixture of varyon-only and IPL-only minidisks, follow the procedure described in "Using Modules at IPL Time" on page 4-55.

To change an IPL-only module to a varyon-only module, follow these steps:

1. Configure the module using the procedure described in "Using an IPL-Only Module" on page 4-58.
2. Use the **minidisks** command to print a list of all the minidisks on the module. Use this list as a check list, to be sure you change all minidisks on the module.
3. Make backup copies of the files **/etc/system** and **/etc/filesystems**. You may need to restore these files from the backup copies if you make mistakes during the rest of the procedure.
4. Edit the file **/etc/system**. To do so, you must have superuser authority.
5. In the file **/etc/system**, edit the stanza for each minidisk on the module. Set the `noipl` attribute to the value `true`, as in this example:

   ```
   noipl = true
   ```

   This setting tells the IPL process that this minidisk is **not** to be included.
6. Edit the file **/etc/filesystems**. To do so, you must have superuser authority.

7. In the file **/etc/filesystems**, edit the stanza for the mount directory corresponding to each minidisk on the module.

   a. Set the following attribute values:

   ```
   mount = false
   check = false
   ```

   b. If you want the **varyon** command to mount the minidisk, set the following attribute value:

   ```
   vmount = true
   ```

   c. If you want the **varyon** command to perform a file consistency check on the minidisk, set the following attribute value:

   ```
   vcheck = true
   ```

To configure the newly-established varyon-only module, follow these steps:

1. Shut down the RT system, using the **shutdown** command.
2. Remove the module from its module location.
3. Start the IPL process by pressing the **Ctrl**, **Alt**, and **Pause** keys at the same time.
4. Insert the module into a module location.
5. Use the **varyon** command to configure the module.

If you made any mistakes during this procedure, and the module is not configured as you wanted, go back to step 4 on page 4-59 and correct the problems.

From now on, configure and unconfigure the module using the **varyon** and **varyoff** commands.

Introduction to International Character Support

The AIX operating system provides international character support in the form of an extended character set that includes accented characters and characters and symbols not used in the English language. A configurable table establishes the character set and the sorting order for the characters. Environment variables let you select alternate time, date, and currency formats, and the characters to use for currency and other symbols. Appropriate defaults can be selected by nationality.

The following terms are defined for use with international character support:

Code page An ordered set of up to 256 characters.

Code point A conceptual character; a 1-byte or 2-byte value that identifies a single character of a code page. AIX translates code points to support character sets for many languages, while remaining compatible with systems and devices that only support ASCII characters.

Collating order The sequence in which characters and character strings are sorted.

Collation table A table that defines the collating order.

Conversion subroutine Provides an appropriate format and strings that the system uses to translate date, time, and currency information for a selected nationality.

Equivalence class A set of characters that are considered equal for purposes of collation. For example, U.S. dictionary collation sequences place an uppercase character in the same equivalence class as its lowercase form, but ASCII collation sorts uppercase and lowercase alphabetical characters into different equivalence classes, because the sort is by the ordinal position of the character. Many collation sequences will not distinguish between accented and unaccented character forms for the purpose of collation, but some will.

Extended character A character other than a 7-bit ASCII character. An extended character can be a 1-byte character with the high-order bit set or a 2-byte character.

Flattened character An extended character translated to an 7-bit ASCII equivalent of similar appearance. For example, the flattened form of è is e.

Features

The international character support provides these features:

Extended characters

Extended characters can be used to support various languages and dialects. Extended characters are supported throughout the system in files, names of files and user, group, and queue names. A few programs do not support extended characters, such as the DOS services shell, and **nroff**.

When a password is defined, the system determines that it is unique, and that its flattened ASCII equivalent is also unique.

The **dd** command and various conversion subroutines translate between extended characters and ASCII escape strings that preserve unique character information. Extended characters can also be converted to ASCII characters of a similar appearance.

Regular expression support

Regular expression support is provided for ASCII and extended characters. Metacharacters retain their traditional use, but now support extended characters.

Configurable Collation

A collating table defines an alphabetical (sort) order, designates character case and sets up equivalence classes. You can alter the collation order, case, and equivalence classes to support a language, dialect, site or user.

String formats

Utilities that output time and date information use a format selected from the environment. The **at** utility accepts input dates in the current format as well.

Numeric symbols

The currency symbol, decimal point character, and thousands-divider character can be set for the site. An example figure using the (U.S.) default is `$9,999.99`.

Configuration

The environment for international character support is established using local environment variables from the user profile. If a variable is not established as a local variable, an **NLFILE** variable is used. If a variable is not defined in the local environment or in **NLFILE**, default values are used. The **NLgetctab** function loads a collation table automatically at system initialization. To change environment settings from an application, **NLgetfile** subroutine runs the **NLgetctab** subroutine. To select an initialization environment suitable for your site, set the environment variable **NLLANG** to the appropriate language string, set the **NLFILE** environment to a suitable environment file and the **NLCTAB** environment variable to the appropriate collation table file name.

A number of environment files are established in directory **/usr/lib/nls**; environment files have an **.en** suffix.

You can revise the collation table file using the **ctab** command, to change the character set, collation sequence, or equivalences values. Collation table files are found in **/usr/lib/nls** directory, and have a **.ctab** suffix. The **ctab** output files in the same directory have no suffix. For example, the output file corresponding to **french.ctab** is **french**.

Code Point Support

The ASCII character set is represented as code points of code page P0, which is a code page composed of 1-byte characters. The 7-bit ASCII characters are lower code page P0. Upper code page P0 is composed of 1-byte extended characters, with the high-order bit set high. Included in code page P0 are ASCII, accented, and other characters that support many languages.

The 1-byte character set is extended by using four values (0x1c, 0x1d, 0x1e, and 0x1f) as single-shift bytes that, together with the next byte in the data stream, select a character from code page P1 or P2. 2-byte characters are handled by an **NLchar** data type. Each **NLchar** occupies two bytes of storage. The high order bit of the **NLchar** is not used in representing a character. An extended character cannot be mistaken for an ASCII character because the high bit is set.

Bytes		Ordinal (NLchar Value)	Ordinal Decimal	Code Page
	0xxxxxxx	000xxxxxxx	0-127	P0 low
	1xxxxxxx	001xxxxxxx	128-255	P0 high
00011111	1xxxxxxx	010xxxxxxx	256-383	P1 low
00011110	1xxxxxxx	011xxxxxxx	384-511	P1 high
00011101	1xxxxxxx	100xxxxxxx	512-639	P2 low
00011100	1xxxxxxx	101xxxxxxx	640-767	P2 high

Figure 4-13. Code Points

Any character in any code page can be entered from the keyboard. Techniques for entering characters that are not supported by the keyboard map are fully described in *Keyboard Description and Character Reference*, Appendix C.

Limits

File and directory names have a size limit of 14 bytes. Because code points can represent either 1 or 2 bytes of storage, the number of significant code points in a file name or directory name can be limited to as few as 7. The kernel checks for a final byte consisting of a single-shift character and replaces that character with a null character.

Intersystems Compatibility

The AIX operating system is compatible with environments that do not support extended characters, providing work station, intersystem mail, networking, and program development support.

An ASCII-character synonym is automatically defined for a login name containing extended characters so that ASCII synonyms do not have to be explicitly maintained in the **/etc/passwd** file. The **adduser** utility checks both the login name containing extended characters and its ASCII equivalent against all existing full IDs, their ASCII equivalents, and synonyms before permitting the login name to be added to the system. Subroutines can convert extended characters to ASCII characters. This provides support for communications to ASCII systems that do not have international character support.

AIX can communicate with systems that use standard ASCII work stations. Site names must conform to the requirements of the host system communication protocol (**SNA**, **uucp**, or other), and must be composed of ASCII characters. Therefore, site names are limited to ASCII characters or some subset of ASCII characters. See *AIX Operating System Communications Guide* for further information.

Environment

To select an environment for international character support, you must provide appropriate environment variables to control date and time strings and to select the collation table. The collation table must be set up appropriately for international character support. For most sites, setting up the environment requires creating an appropriate **NLFILE** value for the collation table file without changing the default collation order or equivalences, and selecting strings and values for time and date conversion. You should set an appropriate **TZ** value to establish time zone information and daylight savings time change points on appropriate Julian dates for the system.

The **vi** editor provides a trusted environment for editing files. For more information on the **vi** editor see *AIX Operating System Commands Reference.*

Time and Date Strings

You can set the string variables for indicating the day of the week, month of the year, and other time strings, such as substitution strings for *yesterday*, *today*, and *noon*. You can set other environment variables as well, for example, the local time value (plus or minus GMT) and a 12-hour or 24-hour time display format.

You can set the system environment with the environment variables described on page 4-65 in the file **/etc/profile**. An environment for an individual can differ from the system environment, and is stored in the **$HOME/.profile** (from the password file) for the user. These variables can also be changed from the command line or by subroutines.

Environment variables for international character support are specified in the process environment using ordinary shell environment variables or in the text file with a path name that is specified by the shell environment variable **NLFILE**. Values specified in the process environment take precedence over values specified in **NLFILE**. If a given environment variable is not set either in the process environment or in **NLFILE**, or if a specified value is the null string, a default value (suitable for U.S. English) is used.

Note: If different international character support environments are in use on the same system, particularly different collation tables or time settings, confusion can result.

The form of environment variables is *variable=value(s).* Environmental variables that establish the local environment vocabulary consist of a sequence of strings separated by colons. The actual language is identified by the value of **NLLANG**. Each string must be a translation of the U.S. English name or symbol used in the defaults, in exactly the same order. You can set the following environment variables for international character support.

NLFILE
Is the path name of a file containing other environment variable definitions for international character support. You cannot define **NLFILE** within a file that is identified by another **NLFILE** definition. There is no default path name.

TZ
Provides time zone strings, the difference in hours from Greenwich Mean Time (GMT), and times for the site to switch to and from daylight savings time. It has eight parameters in five fields in the form:

```
lclndst:bgn:end:chgwd:chghr:chgamt
```

The first field contains the following parameters:

lcl	is the standard local time zone abbreviation.
n	is the difference in local time from GMT in hours (a number from -12 to 12).
dst	is the abbreviation for the local daylight savings time zone, if any.

The remaining fields contain the following:

bgn	is the beginning day (Julian) of daylight savings time, if any.
end	is the ending day (Julian) of daylight savings time, if any.
chgwd	is the weekday of the change to daylight savings time, if any (a numeric value).
chghr	is the hour of the change to daylight savings time, if any (using a 24-hour clock).
chgamt	is the amount, in hours, of the change to daylight savings time.

An example setting is:

```
TZ=EST5.5EDT:119:315:7:2:1.5
```

NLCTAB
Provides the path name of the file containing tables that define the current collating sequence, as produced by **ctab**. The default is:

```
NLCTAB=/etc/nls/ctab/default
```

NLLANG
Provides the language label for the set of strings, collation table, and environment formats used. The default environment label for international character support is:

```
NLLANG=u.s.english
```

NLCURSYM
Provides the currency symbol and symbol placement relative to quantity. Placement is `f` or `F` if the symbol precedes the quantity, and `l` or `L` if the symbol follows the quantity. The default is:

```
NLCURSYM=:$:L:
```

NLNUMSEP
Provides numeric triad and decimal separator symbols. (The first of the two separators is the triad separator.) The defaults are:

```
NLNUMSEP=:,:.:
```

NLLDAY
Provides strings for the full ("long") names for the days of the week. The default values are:

```
NLLDAY=Sunday:Monday:Tuesday:Wednesday:Thursday\
:Friday:Saturday
```

NLLMONTH
Provides the long name strings for the months of the year. The default values are:

```
NLLMONTH=January:February:March:April:May:June:\
July:August:September:October:November:December
```

NLSDAY
Provides short name strings of the days of the week. Names should be the same length, and of 5 or fewer characters. The defaults are:

```
NLSDAY=Sun:Mon:Tue:Wed:Thu:Fri:Sat
```

NLSMONTH
Provides short name strings of the months of the year. Names should be the same length, and of 5 or fewer characters. The default values are:

```
NLSMONTH=Jan:Feb:Mar:Apr:May:Jun:Jul:Aug:Sep:Oct:Nov:Dec
```

NLTMISC
Provides miscellaneous strings needed for input and output of date and time specifications. The default values are:

```
NLTMISC=at:each:every:on:through:am:pm
```

NLTSTRS
Provides strings for relative or informal names used for input of date and time specifications to the **remind** and **at** commands. The default informal time string values are:

```
NLTSTRS=now:yesterday:tomorrow:noon:midnight:\
next:weekdays:weekend
```

NLTUNITS
Provides singular and plural forms of strings for all names of units of time, used for input of date specifications to the **at** command. The defaults are:

```
NLTUNITS=minute:minutes:hour:hours:day:days:\
week:weeks:month:months:year:years
```

NLTIME
Specifies the format of the time. The default time format string is:

```
NLTIME=hh:mm:ss
```

NLDATE
Specifies the short form of the date. The default is:

```
NLDATE=MM/DD/YY
```

NLLDATE
Specifies the long form of the date. The default is:

```
NLLDATE=mon DD, YYYY
```

Collation Table

You can use the **ctab** command to establish collating sequence and case conversion of characters. The input and output files are stored in the conventional directory **/usr/lib/nls**. Numerous files exist to support various user environments, and you can create new ones. (See "Configuration" on page 4-63.) File names should generally reflect their contents — for example, **uk** for British characters, symbols, and lexicographical collating sequence. An example collation table is provided for you in **/usr/lib/nls/example.ctab**.

Using **ctab**, you can use a defined table file for a language without changes, modify an existing table, or build a customized table.

For each character in a collating sequence (each **NLchar**), the table input file provides the following information:

- For alphabetical characters, the corresponding uppercase or lowercase version of the given character. (You can assign case to any character.)
- Collating sequence.
- The range of an equivalence class. All the characters in this class are counted in the character range, along with this character, whenever a character is named as an end point of a character range.
- You can assign a string of characters to a character equivalence class.

The following **ctab** input conventions are used for setting up a table file:

- Input starts from a standard template file containing the entire supported character set in the general order of an ISO collating sequence. Collating sequence follows the ordering of the per-character input lines. Except for ASCII characters and the characters in all supported languages, lines in the file are commented out.

- Escape sequences (in the style of the C language) are permitted in the input table if a backslash does not form part of a valid escape sequence, it strips the following character of any special meaning it could have had.
- One line of information is present for each character explicitly named.

Each character information line in a collation table file contains four data fields:

1. ***Subject Character***
   Identifies the character to appear in the collating order at that point. The treatment of the subject character is dependent on the type of character and its use. The subject character can be:

   - A (nonprinting) control character or a space character.
   - An alphabetical character (having a case equivalent character) and (optional) equivalent characters. If a string of characters is given, they are collated as a single unit.
   - A character to be translated using a defined translation string. The field contains a subject character followed by a | (vertical bar) and a single or multiple-character string; this is a ***translation string*** that the subject character is translated to before collation. For example, to treat the character œ as equivalent to the string oe the line will contain: œ|oe as its first field. However, the subject character cannot be used in the string of characters into which it is to translate. This is an illegal construction:

     o|oe

2. ***case match***
   Identifies the matching case character for the character in field 1 if field 1 is an alphabetical character. If the field 1 character is lowercase, field 2 holds its uppercase equivalent, and visa-versa. If the character in field 1 has no equivalent, this is an empty (null) field.

3. ***type identifier***
   Identifies the type of character in field 1:

   - If the subject character is to be treated as an alphabetic character (even if field 2 is null) this field must identify the character as lowercase or uppercase. An **l** or **L** placed in the type identifier field selects a lowercase subject character (or characters). A **U** or **u** in this field selects an uppercase character (or characters).
   - If the subject character is a (nonprinting) control character, a value of **C** or **c** must be placed in this field.
   - If the subject character is a space (blank or tab) character, a value of **S** or **s** must be placed in this field.

4. ***equivalence class***
   Specifies the first character in the equivalence class of the subject character. Members of an equivalence class must be listed in consecutive lines. If this field is not specified, the group of consecutive subject characters having blank fourth fields are placed in the same equivalence class (only) if they are based on the same roman alphabetic character.

A line beginning with the word `option` serves to change one or more of the default conditions or metacharacters built into **ctab**. This option line contains a set of *name-value* pairs; each pair is delimited by space or tab characters. Recognized *names* are:

eclass
: Turn equivalence class function on or off globally. The *value* must be `on` or `off`; the default is **on**.

sep
: Use the *value* as the separator character between fields on an input line; the default is **:** (colon). Tab or space characters may surround fields and separators.

trans
: Use the *value* as an indicator of translation in subject character fields; the default is **1**.

repeat
: Use the *value* as a repeat indicator meaning *same as previous character* in subject character fields; the default is ^ (circumflex). A repeat character cannot be used following a translation string because the character to be translated has no inherent collating value.

comment
: Use the *value* as a comment character; the default is # (pound sign). That portion of a line to the right of a comment character is ignored.

Terminal Mapping

You can set the terminal maps for your system with the **stty** command. The **imap** parameter sets a terminal **.in** *filename* and the **omap** parameter sets a terminal **.out**. You must select the terminal **.in** and terminal **.out** names from the files in the **/etc/nls/termmap**.

You can establish terminal map assignments in the **/etc/ports** file. The **getty** utility uses this file at system initialization. For further information see the explanation of the **stty** command in AIX Operating System Commands Reference.

Chapter 5. Managing Multiuser Systems

CONTENTS

About This Chapter

This chapter describes the structure, setup, and management of the AIX accounting system, as well as the reports it generates.

This accounting system is designed to provide not only a way to bill for computer use but also a way to monitor various aspects of system operations.

Since you need system data combined and summarized by system users for billing purposes, and you need data combined and summarized by system resources for monitoring system operations, the accounting system collects detailed data on each transaction and provides tools for processing the data to produce different kinds of reports.

Note: The accounting system commands are part of the Multi-User Services licensed program. Before you can use the accounting system, the Multi-User Services licensed program must be installed on your system.

This chapter also describes the design and implementation of the AIX System Activity Package.

Note: The system activity package is part of the Multi-User Services licensed program. Before you can use the system activity package, the Multi-User Services licensed program must be installed on your system.

Running System Accounting

AIX system accounting provides for collecting and processing the following types of system data:

1. The amount of time each user spends logged into the system (***connect-time accounting***).
2. The use by each process of the processing unit, memory, and I/O resources (***process accounting***).
3. The amount of disk space occupied by each user's files.
4. The amount of printer use by each user.
5. Fees charged for materials and services.

Connect-Time Accounting

Data collection

Connect-time data are collected through cooperation between the **init** and **login** commands. When you log in, the **login** program writes a login record in the file **/etc/utmp**. This record includes your user name, the date and time of the login, and the login port. Commands such as **who** use this file to find out which users are logged in and onto which display stations. If the connect accounting file **/usr/adm/wtmp** exists, **login** adds a copy of this login record to it. For more information on **/usr/adm** see "Files and Directories" on page 5-7.

When your login program ends (normally when you log out), the **init** process records the end of the session by writing another record in **/usr/adm/wtmp**. Both the login and logout records have the form described in the header file **utmp.h**. Logout records differ from login records in that they have a blank user name.

The **acctwtmp** command also writes special entries in **/usr/adm/wtmp** having to do with system shutdown and startup.

Reports

The accounting commands concerned with keeping track of each user's share of system resources write standardized ***total accounting records***, whose format is given in the header file **sys/acct.h**. All billable accounting data originates as, or can be converted to, either the total accounting record format or a readable and editable ASCII version thereof.

With the **acctmerg** command, you can convert between the ASCII and binary formats and merge the records from different sources into single records, one for each user.

You can use the **prtacct** command to display any total accounting file. You can also use **awk** scripts to produce site-specific reports from the ASCII representation of the total accounting records. (For a brief discussion of the **awk** command, see *AIX Operating System Commands Reference*. For a more detailed discussion, see *AIX Operating System Programming Tools and Interfaces*.)

The **acctcon1** command produces connect-time and line-use records (***session records***), from login and logout records. From these session records, you can produce reports that show each individual login session, that show a summary of login sessions for each port, or that show an overall summary of login sessions for the system.

You can use the **prctmp** command to display session records, with appropriate headings. The **acctcon2** command converts a collection of session records, sorted by user name, to total accounting records. You usually merge these records with the total accounting records produced by other parts of the accounting system to produce reports for each user.

You can also find out the last date on which each user logged in. The **lastlogin** command produces this report, normally as a part of the **runacct** procedure.

Process Accounting

Data collection

The AIX Operating System collects resource usage data for each process as it runs. The data include the user and group numbers under which the process runs, the first eight characters of the name of the command, the elapsed time and processor time used by the process, memory use, the number of characters transferred, and the number of disk blocks read or written on behalf of the process. The **accton** command causes the kernel to record this data in a specified file, usually **/usr/adm/pacct**. If called without parameters, **accton** turns off the recording of data.

Other commands relating to the recording of process accounting data are **startup**, **shutacct**, **dodisk**, **ckpacct**, and **turnacct**. These commands are simple shell files that build on the other commands discussed in this section.

Reports

In addition to providing billing information, process accounting data also provides a rich source of information that you can use to monitor the use of system resources.

The **acctcms** command summarizes resource use by command name. It provides information on how many times each command was run, how much processor time and memory it used, and how intensive that use was while the command was running (its ***hog factor***). It can also produce long-term statistics on system utilization. Thus, it provides answers to questions about total system usage, what commands are used frequently enough to be worth optimizing, and so on.

The **acctcom** command handles the same data as **acctcms**, but for a different purpose. Whereas **acctcms** summarizes the data by command, **acctcom** provides detailed information about each running of each process. With it, you can display all process accounting records or select records of particular interest. Selection criteria include the load imposed by the process, the time period during which the process ended, the name of the command, the user or group that ran the process, and the port from which the process was run.

The **acctprc1** and **acctprc2** commands produce billing information in two forms. **acctprc1** translates the user ID into a user name and writes ASCII records containing the chargeable items (***prime*** and ***nonprime CPU time***, ***mean memory size***, and ***I/O*** data). **acctprc2** transforms these records into total accounting records.

Disk-Usage Data

Most accounting information on AIX is collected continuously as the resources are consumed. Disk-usage data is an exception. Rather than maintain a running record of disk space consumption, you use the **dodisk** command to collect this data periodically. **dodisk** examines file systems and summarizes space use by user name. You then use **acctmerg** to produce total accounting records from the output of **dodisk**.

The **dodisk** command charges a user name for links to files found under that user's login directory. It divides the charge for a file evenly among links to that file. The decision to charge for links rather than for files owned, is motivated by two considerations:

- The cost of a file should be borne by all who use it. The most reasonable way to determine who uses a file is to see who has links to it.
- A user can create a file, allow another user to link to it and then remove his own link to it. Once he has relinquished his access to the file, it seems unfair to charge him for it.

Printer-Usage Data

Collection of printer-usage data is a cooperative effort between the **print** command that enqueues files to be printed and the queuing daemon that schedules the printing. The **print** command enqueues the user name and number of the user doing the printing, along with the name of the file to be printed and other user-specified flags. After the file has been printed, the **qdaemon** writes to a file, usually **/usr/adm/qacct**, an ASCII record containing the user name, user number, and number of pages printed. You can sort these records, convert them to total accounting records, and use **acctmerg** to combine them with data from other sources.

Fees for Services and Materials

You use the **chargefee** command to charge users for services, such as file restores or consulting, or for certain materials. **chargefee** writes an ASCII total accounting record in the file **/usr/adm/fee**.

Files and Directories

The directory **/usr/lib/acct** contains all of the C Language programs and shell procedures necessary to run AIX system accounting. The accounting system data files belong to user **adm** (currently user ID 4), and all active data files (**wtmp**, **pacct**, etc.), reside in this user's home directory, **/usr/adm**. All reports and summary files are stored in the subdirectories **nite**, **sum**, and **fiscal** (see Figure 5-1).

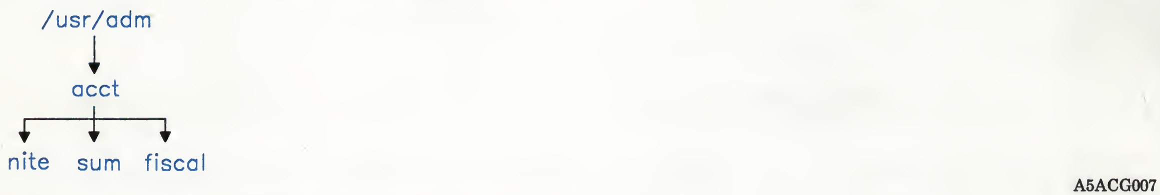


Figure 5-1. Accounting Directory Structure

The **nite** directory contains files that **runacct** reuses daily. The **sum** directory contains the cumulative summary files that **runacct** updates. Finally, the **fiscal** directory contains the summary files that **monacct** creates.

Setting Up the Accounting System

In order to automate the operation of this accounting system, you need to do several things.

To Set up System Accounting

1. Enter:

   `/usr/lib/acct/nulladm wtmp pacct`
2. Update the file **/usr/lib/acct/holidays**.
3. Turn on process accounting through the file **/etc/rc.include**.
4. In the file **/etc/filesystems**, identify file systems to include in disk accounting.
5. Specify the data file for printer usage data in the file **/etc/qconfig**.
6. Schedule daily accounting in the file **/usr/spool/cron/crontab/adm**.
7. Schedule monthly or fiscal summaries in the file **/usr/spool/cron/crontab/adm**.
8. Set the PATH shell variable in **/usr/adm/def.profile** to include **/usr/lib/acct**.

Accounting Setup Procedure

1. Run the **nulladm** procedure as follows:
   a. Log in as user **root**.
   b. Enter:

      `/usr/lib/acct/nulladm wtmp pacct`

   The **nulladm** procedure ensures that each file has the proper access permission code (664).
2. Update the file **/usr/lib/acct/holidays**, if necessary. This file contains the time table that the accounting system uses to distinguish prime time and nonprime time. Edit this table to reflect your holiday schedule for the year and to reflect the hours during each day that you want to designate as prime time.

   This file contains three types of entries:
   a. Comment lines:

      Comment lines may appear anywhere in the file as long as the first character in the line is an asterisk (*).

b. Year Designation Line:

   This line must be the first data line (that is, the first line that is not a comment) and may appear only once. It consists of three fields of four digits each (leading blanks ignored). These three fields contain:

   1) The year
   2) The time (*hhmm*) at which prime time begins
   3) The time (*hhmm*) at which prime time ends.

   Use a 24-hour clock to specify the time. You can enter the hour of midnight as either 0000 or 2400.

   For example, to specify the year as 1984, prime time as beginning at 8:00 a.m. and nonprime time as beginning at 5:00 p.m., enter the following line:

```
1984    0800    1700
```

c. Company Holiday Lines:

   These entries come after the year designation line. Each of these lines, composed of four fields, has the following general format:

```
Year-day  Month  Day  Description of Holiday
```

   The Year-day field contains a number from 1 through 366 that indicates the day of the year on which the holiday falls (leading blanks ignored). The other three fields—Month, Day, and Description of Holiday— are treated as comments by other accounting programs. These fields identify to other users what holiday is designated by the number in the Year-Day field. Figure 5-2 on page 5-10 shows a sample **holidays** file.

```
* Curr  Prime  Non-Prime
* Year  Start  Start
*
  1985  0830   1700
*
* Day of    Calendar    Company
* Year      Date        Holiday
*
    1       Jan 1       New Year's
    2       Jan 2       day after
   46       Feb 15      Wash. Birthday
  152       May 31      Memorial Day
  186       Jul 5       Indep. Day
  249       Sep 6       Labor Day
  332       Nov 28      Thanksgiving
  333       Nov 29      day after
  358       Dec 24      Christmas eve
```

Figure 5-2. Sample holidays file

3. To turn on process accounting, add the following line to **/etc/rc**, if it is not already there (or delete the comment symbol if it is present):

   ```
   /bin/su - adm -c /usr/lib/acct/startup
   ```

   The **startup** shell procedure calls **acctwtmp** to record in **/usr/adm/wtmp** the time that accounting was turned on. It calls **turnacct on**, which cleans up the previous day's accounting files by calling the **remove** shell procedure.

   The **remove** procedure deletes all **/usr/adm/acct/sum/wtmp***, **/usr/adm/acct/sum/pacct***, and **/usr/adm/acct/nite/lock*** files.

4. In **/etc/filesystems**, add the following line to each stanza that defines a file system you want to include in system accounting:

   ```
   account = true
   ```

5. Enable printer usage accounting by adding the following line to the queue stanza in the file **/etc/qconfig**:

   ```
   acctfile = /usr/adm/qacct
   ```

6. To automatically run daily accounting through **cron**, add the following lines to the file **/usr/spool/cron/crontab/adm**. (See the discussions of **cron** and **crontab** in *AIX Operating System Commands Reference* for details about submitting jobs to **cron**.) If these times do not fit the hours that your system is operational, adjust your entries accordingly.

```
0 2 * * 4 /usr/lib/acct/dodisk
5 * * * * /usr/lib/acct/ckpacct
0 4 * * 1-6 /usr/lib/acct/runacct
              2>/usr/adm/acct/nite/accterr
```

The first entry schedules the running of **dodisk** for 2:00 a.m. (0 2), each Thursday (4). This second schedules **ckpacct** for 5 minutes past every hour (5 *) of every day (*). The third schedules **runacct** for 4:00 a.m. (0 4) every Monday through Saturday (1-6).

The **dodisk** procedure calls **diskusg** and **acctdisk** to write disk usage records to the file **dacct**. It stores this file in **/usr/adm/acct/nite**.

The **ckpacct** procedure monitors the size of **/usr/adm/pacct**. If the file is larger than 500 blocks, **ckpacct** calls **turnacct switch** to copy the current **pacct** file to **pacct***x*, where *x* is an integer that is increased each time **turnacct switch** is called. The advantage of having several smaller **pacct** files becomes apparent when you must restart **runacct** after a failure in processing these records.

The **runacct** procedure processes the active data files (including **wtmp**, **pacct**) to produce command summaries and usage summaries sorted by user name. For a detailed discussion of **runacct**, see "Running Daily Accounting—The runacct Command" on page 5-12.

7. To automatically perform monthly merging of accounting data, add the following line to the file **/usr/spool/cron/crontab/adm**:

   ```
   15 5 1 * * /usr/lib/acct monacct
   ```

   Be sure to schedule this procedure at a time (5:15 a.m. here), that allows **runacct** sufficient time to finish. This will, on the first day of each month, create monthly accounting files with the entire month's data. For a detailed discussion of **monacct** reports, see "Daily Command and Monthly Total Command Summaries" on page 5-18.

8. Set the PATH shell variable in **/usr/adm/.profile** as follows:

   ```
   PATH=/usr/lib/acct:/bin:/etc:/usr/bin::
   export PATH
   ```

   This simplifies the running of individual accounting commands when you are logged in as **adm**.

Running Daily Accounting—The runacct Command

The **runacct** command is the main daily accounting shell procedure. Normally initiated by **cron** during nonprime hours, **runacct** processes connect, fee, disk, and process accounting files. It also prepares daily and cumulative summary files for use by the **prdaily** command or for billing purposes.

Output Files

The following files produced by **runacct** are of particular interest. See Figure 5-3 on page 5-19 for a description of Accounting file formats.

/usr/adm/acct/nite/lineuse
: This file contains usage statistics for each terminal line on the system. This report is especially useful for detecting bad lines. If the ratio between the number of logouts and logins exceeds about 3 to 1, there is a good possibility that a line is failing.

/usr/adm/acct/nite/daytacct
: This is the total accounting file for the previous day.

/usr/adm/acct/sum/tacct
: This file contains the accumulation of each day's **nite/daytacct** file and can be used for billing purposes. **monacct** restarts it each month or fiscal period.

/usr/adm/acct/sum/cms
: This file contains the accumulation of each day's command summaries. **monacct** uses this binary version of the file and restarts it. The ASCII version is **nite/cms**.

/usr/adm/acct/sum/daycms
: This file contains the daily command summary. An ASCII version is stored in **nite/daycms**.

/usr/adm/acct/sum/loginlog
: This file contains a record of the last time each user ID was used.

/usr/adm/acct/sum/rprt*mmdd*
: This file contains a copy of the daily report saved by **runacct**.

Operational States

The **runacct** procedure takes care to protect active data files. It checks frequently for operational errors. If it detects one, it writes a message to **/dev/console** (the command line in **crontab/adm** should redirect these messages to the file **/usr/adm/acct/nite/accterr**), mails messages to the users root and **adm**, removes locks, saves diagnostic files, and exits.

To make **runacct** easier to restart after an error, its operation is divided into clearly defined stages, or ***operational states***. It records its progress through these states by writing descriptive messages in the file **/usr/adm/acct/nite/active**. It also writes its current state to the file **/usr/adm/acct/nite/statefile**. The last operation in each state is to write the next state to **statefile**. After it completes each state, **runacct** reads **statefile** for the next state to process.

These states are processed as follows:

State	Actions
SETUP	This state calls **turnacct switch**. It moves the process accounting files **/usr/adm/pacct***?* to **/usr/adm/Spacct***?.mmdd*. It also calls **acctwtmp** to write the current time as the last record in **/usr/adm/wtmp** and then moves the file to **/usr/adm/acct/nite/wtmp***.mmdd*.
WTMPFIX	This state calls **wtmpfix** to check **nite/wtmp** for accuracy. Some date changes cause problems with the **acctcon1** command, so **wtmpfix** attempts to adjust the time stamps in the **wtmp** file if a date change record appears.
CONNECT1	This state calls **acctcon1** to write session records from **wtmp** to the **ctmp** file (see Figure 5-3 on page 5-19). It also creates the **lineuse** file and the **reboots** file. Together, these three files contain all of the login and logout records found in the **wtmp** file.
CONNECT2	This state calls **acctcon2** to convert **ctmp** files to total accounting files. It sends these to **acctmerg** to produce the **ctacct***.mmdd* file. This file contains all the connect accounting records (see Figure 5-3 on page 5-19).
PROCESS	This state calls **acctprc1** and **acctprc2** to convert the process accounting files **/usr/adm/Spacct***?.mmdd* into total accounting records in **ptacct***?.mmdd* (see Figure 5-3 on page 5-19). The **Spacct** and **ptacct** files are correlated by number so that if **runacct** fails in this state, you do not need to reprocess all **Spacct** files. **Note:** When restarting **runacct** in the PROCESS state, remove the last **ptacct** file as it will not be complete.
MERGE	This state calls **acctmerg** to merge all process accounting records with the connect accounting records, writing these to the **daytacct** file (see Figure 5-3 on page 5-19).

FEES	This state calls **acctmerg** to merge any records from the file **fee** into **daytacct**.
DISK	This state calls **acctmerg** to merge **dacct** with **daytacct**.
QUEUEACCT	This state sorts the queue (printer) usage records, converts them into total accounting records, and calls **acctmerg** to merge them with the other total accounting records in **daytacct**.
MERGETACCT	This state calls **acctmerg** to merge **daytacct** with **sum/tacct**, the cumulative total accounting file. Each day, **daytacct** is saved as **sum/tacct***mmdd*, so that **sum/tacct** can be recreated in the event it becomes damaged or lost.
CMS	This state merges the current day's command summary with the cumulative command summary file **sum/cms**, producing both ASCII and binary summary files (see Figure 5-3 on page 5-19).
USEREXIT	If the shell file **/usr/adm/siteacct** exists, this state calls it to perform site-dependant processing.
CLEANUP	This state cleans up temporary files, runs **prdaily** and saves its output in **sum/rprt***mmdd*, removes the locks, and exits.

Recovering From Failure

The **runacct** procedure can fail for a variety of reasons, most commonly because the system goes down, the file system **/usr** runs out of space, or the **wtmp** file has records with inconsistent date stamps. If **runacct** fails, do the following:

1. Check the file **/usr/adm/acct/nite/active***mmdd* first for error messages.
2. If both the **active** file and lock files exist in acct/nite, check the file **accterr** (the file you redirected error messages to when **cron** called **runacct**).
3. Perform any actions needed to eliminate errors (see "Fixing Damaged Files" on page 5-15).
4. Restart **runacct** (see the following section, "Restarting runacct").

Restarting runacct

If you call **runacct** with no command line parameters, it assumes that this is the first invocation of the day. You need to include the parameter *mmdd* if you are restarting **runacct**. This parameter specifies the month and day for which **runacct** is to rerun accounting. If you do not specify a *state*, **runacct** determines the entry point for processing by reading **statefile**. To override **statefile**, specify the desired *state* on the command line.

Note: When you perform the following tasks, you may need to use the full path name of **runacct** (**/usr/lib/acct/runacct**) rather than the simple command name.

1. To start **runacct**, enter:

   ```
   nohup runacct 2>/usr/adm/acct/nite/accterr &
   ```

   This entry causes **runacct** to ignore all **INTERRUPT** and **QUIT WITH DUMP** signals while it performs its processing in the background. It also redirects all standard error output (file descriptor 2) to the file **/usr/adm/acct/nite/accterr**.

2. To restart **runacct**:

   ```
   nohup runacct 0601 2>>/usr/adm/acct/nite/accterr &
   ```

   This restarts **runacct** for day of June 1 (0601). **runacct** reads the file **/usr/adm/acct/nite/statefile** to find out which state it should begin with. All standard error output is appended to the file **/usr/adm/acct/nite/accterr**.

3. To restart **runacct** at a specified state, in this case the MERGE state, enter the following:

   ```
   nohup runacct 0601 MERGE 2>>/usr/adm/acct/nite/accterr &
   ```

Fixing Damaged Files

Unfortunately, a file occasionally becomes damaged or lost. Certain files must be fixed in order to maintain the integrity of the accounting system.

Fixing wtmp *Errors*

The **wtmp** files seem to cause the most problems in the day-to-day operation of the accounting system. When the date is changed and the AIX Operating System is in multiuser mode, a set of date change records is written to **/usr/adm/wtmp**. The **wtmpfix** command is designed to adjust the time stamps in the **wtmp** records when a date change is encountered. However, some combinations of date changes and system restarts will slip through **wtmpfix** and cause **acctcon1** to fail. The following steps show how to patch up a **wtmp** file:

```
cd  /usr/adm/acct/nite
fwtmp  <wtmp.mmdd >wtmp.new
ed  wtmp.new
```

(Delete damaged records or
delete all records from beginning
up to the date change)

```
fwtmp  -ic   <wtmp.new  >wtmp.mmdd
```

The **fwtmp** command without a flag converts a binary **utmp** file to an ASCII file that you can edit. **fwtmp** with the **-ic** flag converts the ASCII file back to the binary **utmp** format.

If the **wtmp** file is beyond repair, use the **nulladm** command to create an empty **wtmp** file. This prevents any charging of connect time.

Fixing tacct *Errors*

If you are using the accounting system to charge users for system resources, the integrity of **sum/tacct** is quite important. Occasionally, mysterious **tacct** records appear that contain negative numbers, duplicate user numbers, or a user number of 65,535. If this happens, first use the **prtacct** command to check the **sum/tacctprev** file. If it looks all right, the latest **sum/tacct**.*mmdd* should be patched up, then **sum/tacct** recreated. A simple patchup procedure would be:

```
cd  /usr/adm/acct/sum
acctmerg -v  <tacct.mmdd  >tacct.new
ed tacct.new
```

(Remove the bad records.
Write duplicate user number
records to another file.)

```
acctmerg -i  <tacct.new  >tacct.mmdd
acctmerg tacctprev  <tacct.mmdd >tacct
```

Remember that the **monacct** procedure removes all the **tacct**.*mmdd* files; therefore, **sum/tacct** can be recreated by merging these files together.

Accounting Reports

The **runacct** procedure produces five basic reports. They cover the areas of connect accounting, daily use of system resources by each user, command usage reported by daily and monthly totals, and a report of the last time users were logged in.

The following sections describe the reports and the meaning of their tabulated data.

Daily Report

In the first part of the **/usr/adm/acct/sum/rprt** *mmdd* report, a `from . . . to` banner identifies the period reported on. The beginning time for each report is the time the last accounting report was produced. The ending time is the time the current accounting report was produced. The banner is followed by a log of system restarts, shutdowns, power fail recoveries, and any other record written to **/usr/adm/wtmp** by **acctwtmp**.

The second part of the report breaks down line use. The `TOTAL DURATION` entry tells how long the system was in the multiuser state (that is, able to be accessed through terminal lines).

The report columns are:

LINE	The terminal line or access port.
MINUTES	The total number of minutes that line was in use during the accounting period.
PERCENT	The total number of MINUTES the line was in use divided into the TOTAL DURATION.
# SESS	The number of times this port was accessed for a login session.
# ON	The same as # SESS.
# OFF	The total number of both logouts and interrupts that occur on a line.

During real time, you should monitor **/usr/adm/wtmp**. If it grows rapidly, run **acctcon1** to see which line is the noisiest. System performance is affected as the interrupt rate increases.

Daily Usage Report

This report, **/usr/adm/acct/nite/lineuse**, gives a breakdown of system resource use by user. Its data consists of:

UID	The user ID.
LOGIN NAME	The user name of the user. There can be more than one user name for a single user ID; this identifies which one is referred to.
CPU (MINS)	The amount of time that the user's process used the central processing unit. This category is broken down into PRIME and NPRIME (nonprime) use. The accounting system's definition of prime and nonprime time is taken from the **/usr/lib/acct/holidays** file.
KCORE-MINS	A cumulative measure of the amount of memory a process uses while running. The amount shown reflects kilobyte segments of memory used per minute. This measurement is also broken down into PRIME and NPRIME amounts.
CONNECT (MINS)	The amount of time that a user was logged into the system. This column is also subdivided into PRIME and NPRIME use.
DISK BLOCKS	The number of disk blocks used.
# OF PROCS	The number of processes that were invoked by the user. This is a good column to watch for large numbers indicating that a user may have a faulty shell procedure.
# OF SESS	The number of times the user logged onto the system.

DISK SAMPLES — The number of times the disk accounting was run to obtain the average number of DISK BLOCKS listed earlier.

FEE — The total charged against the user by the **chargefee** command.

Daily Command and Monthly Total Command Summaries

These two reports, **/usr/adm/acct/sum/cms** and **/usr/adm/acct/sum/daycms**, are virtually the same except that the Daily Command Summary only reports on the current accounting period while the Monthly Total Command Summary tells the story for the start of the fiscal period to the current date. In other words, the monthly report reflects the data accumulated since the last running of **monacct**.

These reports are sorted by TOTAL KCOREMIN, which is an arbitrary yardstick but often a good one for calculating drain on a system

COMMAND NAME — The name of the command. Unfortunately, all shell procedures are lumped together under the name **sh** since only object modules are reported by the process accounting system. You should monitor the frequency of programs called **a.out** or **core** or any other name that does not seem quite right. **acctcom** is also a good tool to use to determine who ran a specific command and to determine if superuser authority was used.

NUMBER CMDS — The total number of invocations of this particular command.

TOTAL KCOREMIN — The total cumulative measurement of the amount of kilobyte segments of memory used by a process per minute of run time.

TOTAL CPU-MIN — The total processing time this program has accumulated.

TOTAL REAL-MIN — The total ***real-time*** (wall clock) minutes this program has accumulated. This total is the actual time that you must wait for a system prompt, as opposed to starting a process in the background.

MEAN SIZE-K — This is the mean of the TOTAL KCOREMIN over the number of invocations reflected by NUMBER CMDS.

HOG FACTOR — A relative measurement of the ratio of system availability to system utilization. It is computed by the formula:

(total processor time) / (elapsed time)

This gives a relative measure of the total available processor time consumed by the process during its execution.

CHARS TRNSFD	The total count of the number of characters transferred by the **read** and **write** system calls. (This total may be a negative number.)
BLOCKS READ	A total count of the physical block reads and writes that a process performed.

Last Login

This report, **/usr/adm/acct/sum/loginlog**, simply gives the date when a particular user name was last used. This could be a good source for finding likely candidates for the archives or for getting rid of unused logins and login directories.

Accounting File Formats

Figure 5-3 provides a description of the formats of the major accounting data and summary data files.

File Name	Record Format	Contents
wtmp	utmp.h	Login records: 1. user number 2. user name 3. device name 4. process ID 5. entry type 6. exit status 7. date / time
ctmp	ASCII	Connect records: 1. user number 2. user name 3. device name 4. prime connect time 5. nonprime connect time 6. session starting time 7. starting date

Figure 5-3 (Part 1 of 3). Accounting File Formats

File Name	Record Format	Contents
pacct* **Spacct***	sys/acct.h	Process records: 1. beginning time 2. user time 3. system time 4. elapsed time 5. memory used 6. chars transferred 7. blocks read / written 8. command name
daytacct **sum/tacct**	tacct (sys/acct.h)	Total accounting records: 1. user number 2. user name 3. total kcore minutes 4. total connect time 5. total disk usage 6. number of processes 7. number of login sessions 8. number of disk samples 9. fees for special services 10. total chars transferred 11. total blocks read / written 12. queuing system charges
ptacct	tacct	Process total accounting records: 1. user number 2. user name 3. total kcore minutes 4. number of processes 5. total chars processed 6. total blocks read / written
ctacct	tacct	Connect total accounting records: 1. user number 2. user name 3. total connect time

Figure 5-3 (Part 2 of 3). Accounting File Formats

File Name	Record Format	Contents
cms daycms	binary & ASCII	Command summary records: 1. command name 2. number of times called 3. total kcore minutes 4. total processor minutes 5. total read minutes 6. mean memory size 7. mean processor size 8. Hog factor

Figure 5-3 (Part 3 of 3). Accounting File Formats

Accounting System Files

Files in the /usr/adm Directory

diskdiag Diagnostic output during the execution of disk accounting programs.

dtmp Output from the **acctdusg** command.

fee Output from the **chargefee** command, in ASCII **tacct** records.

pacct Active process accounting file.

Spacct?.*MMDD* Process accounting files for *MMDD* during the execution of **runacct**.

Files in the /usr/adm/acct/nite Directory

active Used by **runacct** to record progress and print warning and error messages. The file **active*MMDD*** is a copy of **active** made by **runacct** after it detects an error.

cms ASCII total command summary used by the **prdaily** command.

ctacct.*MMDD* Connect total accounting records.

ctmp Connect session records.

daycms ASCII daily command summary used by the **prdaily** command.

daytacct Total accounting records for one day.

dacct Disk total accounting records, created by **dodisk**.

accterr	Diagnostic output produced during the execution of **runacct**.
lastdate	The last day **runacct** executed, in **date +%m%d** format.
lock **lock1**	Used to control serial use of **runacct**.
lineuse	tty line usage report used by **prdaily**.
log	Diagnostic output from **acctcon1**.
log*mmdd*	Same as **log** after **runacct** detects an error.
reboots	Contains beginning and ending dates from **wtmp**, and a listing of system restarts.
statefile	Used to record the current state during execution of **runacct**.
tmpwtmp	wtmp file corrected by **wtmpfix**.
wtmperror	Contains **wtmpfix** error messages.
wtmperr*mmdd*	Same as **wtmperror** after **runacct** detects an error.
wtmp.*mmdd*	The previous day's **wtmp** file.

Files in the /usr/adm/acct/sum Directory

cms	Total command summary file for the current fiscal period, in binary format.
cmsprev	The command summary file without the latest update.
daycms	The command summary file for the previous day, in binary format.
lastlogin	The file created by **lastlogin**.
pacct.*mmdd*	Concatenated version of all **pacct** files for *mmdd*. This file is removed after system startup by the **remove** procedure.
rprt*mmdd*	The saved output of **prdaily**.
tacct	The cumulative total accounting file for the current fiscal period.
tacctprev	The same as **tacct** without the latest update.
tacct*mmdd*	The total accounting file for *mmdd*.
wtmp.*mmdd*	The saved copy of the **wtmp** file for *mmdd*. This file is removed after system startup by the **remove** procedure.

Files in the /usr/adm/acct/fiscal Directory

cms*?*	The total command summary file for fiscal period *?*, in binary format.
fiscrpt*?*	A report similar to that of **prdaily** for fiscal period *?*.
tacct*?*	The total accounting file for fiscal period *?*.

Using the System Activity Package

The AIX Operating System contains a number of counters that are incremented as various system actions occur. The system activity package reports system-wide measurements, including central processing unit utilization, disk and tape I/O activities, terminal device activity, buffer usage, system calls, system switching, file-access activity, queue activity, and message and semaphore activities.

The package provides three commands that generate various types of reports. Procedures that automatically generate daily reports are also included. The four functions of the activity package are:

The **sar** command
: This command allows you to generate system activity reports in real-time and to save system activities in a file for later use.

The **sag** command
: This command displays system activity in a graphical form.

The **timex** command
: This command is a modified **time** command that times a process and also optionally reports concurrent system activity and process accounting activity.

Daily Reports
: Procedures are provided for sampling and saving system activities in a data file periodically and for generating the daily report from the data file.

The system activity information reported by this package is derived from a set of system counters located in the kernel. These system counters are described under "System Activity Counters." "System Activity Commands" on page 5-27 describes the commands provided by this package. The procedure for producing daily reports is described under "System Activity Daily Reports" on page 5-28.

System Activity Counters

The AIX Operating System manages a number of counters that record various activities and provide the basis for the system activity reporting system. The data structure for most of these counters is defined in the **sysinfo** structure in **/usr/include/sys/sysinfo.h** (see "System Activity Data Structures and File Formats" on page 5-29). The system table overflow counters are kept in the **syserr** structure. The device activity counters are extracted from the device status tables.

The following is a list of the system activity counters sampled by the system activity package:

CPU time counters
At each clock interrupt, the system increments one of four time counters (**cpu[]**), depending on the mode the CPU is in at the interrupt (idle, user, kernel, and wait for I/O completion).

lread and lwrite
The **lread** and **lwrite** counters contain the count of logical read and write requests issued by the system to block devices.

bread and bwrite
The **bread** and **bwrite** counters contain a count of the number of times data are transferred between the system buffers and the block devices. The ratio of block I/O to logical I/O is a common measure of the effectiveness of system buffering.

phread and phwrite
The **phread** and **phwrite** contains a count of the read and write requests issued by the system to raw devices.

pswitch and syscall
These counters are related to the management of multiprogramming. **syscall** is incremented every time a system call is invoked. The numbers of invocations of **read**, **write**, **fork**, and **exec** system calls are kept in counters **sysread**, **syswrite**, **sysfork**, and **sysexec**. **pswitch** counts the times the *switcher* was invoked, which occurs when:

1. A system call resulted in a road block.
2. An interrupt occurred resulting in awakening a higher priority process.
3. A one-second clock interrupt occurs.

iget, namei, and dirblk
These counters apply to file-access operations. **iget** and **namei**, in particular, are the names of AIX Operating System routines. The counters record the number of times the respective routines are called.

The **namei** routine performs file system path searches. It searches the various directory files to get the associated i-number of a file corresponding to a special file.

The **iget** routine locates the i-node entry of a file. It first searches the i-node table in memory. If the i-node entry is not in the table, the **iget** routine gets the i-node from the file system where the file resides and enters it in the i-node table in memory. **iget** returns a pointer to this entry.

The **namei** routine calls **iget**, but other file access routines also call **iget**. Therefore, counter **iget** is always greater than counter **namei**.

Counter **dirblk** records the number of directory block reads issued by the system. Note that the directory blocks read divided by the number of **namei** calls estimates the average path length of files.

runque and runocc

These counters record queue activities. They are implemented in the **clock.c** routine. The clock routine periodically examines the process table to see whether any processes are in memory and in ready state. If so, the system increments the counter **runocc** and adds the number of such processes to counter **runque**.

readch and writech

The **readch** and **writech** counters record the total number of bytes (characters) transferred by the **read** and **write** system calls.

Monitoring terminal device activities

There are six counters monitoring terminal device activities. **rcvint**, **xmtint**, and **mdmint** are counters measuring hardware interrupt occurrences for receiver, transmitter, and modem individually. **rawch**, **canch**, and **outch** count the number of characters in the raw queue, canonical queue, and output queue. Characters generated by devices operating in the "cooked" mode, such as terminals, are counted in both **rawch** and (as edited) in **canch**; but characters from raw devices, such as communication processors, are counted only in **rawch**.

msg and sema counters

These counters record message sending and receiving and semaphore operations.

Monitoring I/O activities

Four counters are kept for each disk or tape drive in the device status table. Counter `io_ops` is incremented when an I/O operation has occurred on the device. It includes block I/O, and physical I/O. `io_bcnt` counts the amount of data transferred between the device and memory in 512-byte units. `io_act` and `io_resp` measure the active time and response time for a device in time ticks summed over all I/O requests that have completed for each device. The device active time includes the device seeking, rotating, and data transferring times, while the response time of an I/O operation is the time the I/O request is queued to the device to the time when the I/O completes.

i-nodeovf, fileovf, textovf, and procovf

These counters are extracted from the `_syserr` structure. When an overflow occurs in any of the i-node, file, text, and process tables, the corresponding overflow counter is incremented.

System Activity Commands

The system activity package provides three commands for generating various system activity reports and one command for profiling disk activities. These tools facilitate observation of system activity during:

- A controlled standalone test of a large system
- An uncontrolled run of a program to observe the operating environment
- Normal production operation.

The **sar** and **sag** commands permit you to specify a sampling interval and number of intervals for examining system activity and then to display the observed level of activity in tabular or graphical form. The **timex** command reports the amount of system activity that occurred during the precise period of execution of a timed command.

The sar Command

The **sar** command can be used in the following two ways:

- When you specify the frequency parameters *interval* and *number*, **sar** invokes the data collection program **sadc** to sample the system activity counters in the operating system every *interval* seconds for *number* intervals and generates system activity reports in real-time. Generally, it is desirable to include the option to save the sampled data in a file for later examination. For the format of this file, see "System Activity Data Structures and File Formats" on page 5-29. In addition to the system counters, a time stamp is also included. It gives the time at which the sample was taken.
- If you do not supply frequency parameters, **sar** generates system activity reports for a specified time interval from an existing data file that was created by **sar** at an earlier time.

A convenient usage is to run **sar** as a background process, saving its samples in a temporary file but sending its standard output to **/dev/null**. Then an experiment is conducted after which the system activity is extracted from the temporary file.

See **sar** in *AIX Operating System Commands Reference* for a discussion of all flags and usage.

The sag Command

The **sag** command displays system activity data graphically. It relies on the data file produced by a prior run of **sar**, from which any column of data or the combination of columns of data can be plotted. A fairly simple but powerful command syntax allows the specification of cross plots or time plots. Data items are selected using the **sar** column header names.

See **sag** in *AIX Operating System Commands Reference* for a discussion of all flags and usage.

The timex Command

The **timex** command is an extension of the **time** command. Without flags, **timex** behaves like **time**. In addition to giving the time information, it can also display a system activity report and a process accounting report. For all the flags available, see *AIX Operating System Commands Reference.*

In the report, the **user** and **sys** times reported in the second and third lines apply to the measured process itself, including all its children. The remaining data, including **cpu user %** and **cpu sys %**, apply to the entire system.

While the normal use of **timex** will probably be to measure a single command, multiple commands can also be timed, either by combining them in an executable file and timing it or by entering:

```
timex sh -c "cmd1; cmd2;...;"
```

This establishes the necessary parent-child relationships to correctly extract the user and system consumed by *cmd1*, *cmd2*, . . . (and the shell).

System Activity Daily Reports

The previous section described the commands available to users to initiate activity observations. It is probably desirable for each installation to routinely monitor and record system activity in a standard way for historical analysis. This section describes the steps that you may follow to automatically produce a standard daily report of system activity.

Facilities

sadc This command reads system counters from **/dev/kmem** and records then in a file. In addition to the file parameter, two frequency parameters are usually specified to indicate the sampling interval and number of samples to be taken. If frequency parameters are not given, it writes a dummy record in the file to indicate s system restart.

sa1 The shell procedure that invokes **sadc** to write system counters in the daily data file **/usr/adm/sa/sa***dd*, where *dd* represents the day of the month. It may be invoked with sampling interval and iterations as parameters.

sa2 The shell procedure that invokes **sar** to generate daily report **/usr/adm/sa/sar***dd* from the daily data file **/usr/adm/sa/sa***dd.* It also removes the daily data files and report files after seven days. The starting and ending times and all report options of **sar** are applicable to **sa2**.

Suggested Operational Setup

You should use **cron** to control the normal data collection and report generation operations. For example, the sample entries in **/usr/spool/cron/crontabs/sys**:

```
0 * * * 0,6 /usr/lib/sa/sa1
0 18-7 * * 1-5 /usr/lib/sa/sa1
0 8-17 * * 1-5 /usr/lib/sa sa1 1200 3
```

would cause the data collection program **sadc** to be invoked every hour on the hour. Moreover, depending on the arguments presented, it writes data to the data file one to three times at every 20 minutes. Therefore, under the control of **cron**, the data file is written every 20 minutes between 8:00 and 18:00 on weekdays and hourly at other times.

Note that data samples are taken more frequently during prime time on weekdays to make them available for a finer and more detailed graphical display. You should invoke **sa1** hourly rather than invoking it once every day, as this ensures that if the system crashes, data collection will be resumed within an hour after the system is restarted.

Because system activity counters restart from zero when the system is restarted, a special record is written on the data file to reflect this situation. This process is accomplished by invoking **sadc** in **/etc/rc.include**, without frequency parameters, when going to multiuser state:

```
/usr/lib/sa/sadc /usr/adm/sa/sa` 'date +d%`'
```

cron also controls the invocation of **sar** to generate the daily report via shell procedure **sa2**. You may choose the time period the daily report is to cover and the groups of system activity to be reported. For instance, if:

```
0 20 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:00 -i 3600 -uybd
```

is an entry in **/usr/spool/cron/crontabs/sys**, **cron** will execute the **sar** command to generate daily reports from the daily data file at 20:00 on weekdays. The daily report reports the processor utilization, terminal device activity, buffer usage, and device activity every hour from 8:00 to 18:00.

In case of a shortage of the disk space or for any other reason, these data files and report files can be removed by a user with superuser authority.

System Activity Data Structures and File Formats

This section contains the data structures and file formats used by the system activity package.

sysinfo.h

```
#ifndef _h_SYSINFO
#define _h_SYSINFO

struct sysinfo {
        time_t      cpu[4]
#define CPU_IDLE   0
#define CPU_USER  1
#define CPU_KERNEL     2
#define CPU_WAIT  3
        time_t      wait[3]
#define W_IO   0
#define W_SWAP     1
#define W_PIO 2
        long bread;
        long bwrite;
        long lread;
        long lwrite;
        long phread;
        long phwrite;
        long pswitch;
        long syscall;
        long sysread;
        long syswrite;
        long sysfork;
        long sysexec;
        long runque;
        long runocc;
        long iget;
        long namei;
        long dirblk;
        long readch;
        long writech;
        long rcvint;
        long xmtint;
        long mdmint;
        long rawch;
        long canch;
        long outch;
        long msg;
        long sema;
        long ksched;
        long koverf;
        long kexit;
        long rbread;
        long rcread;
```

```
        long rbwrt;
        long rcwrt;
};

extern struct sysinfo sysinfo;

struct syswait {
        short       iowait;
        short       physio;
};

extern struct syswait syswait;

syserr {
        long inodeovf;
        long fileovf;
        long textovf;
        long procovf;
        long sbi[5];
#define SBI_SILOC  0
#define SBI_CRDRDS      1
#define SBI_ALERT 2
#define SBI_FAULT 3
#define SBI_TIMEO 4
};

extern struct syserr syserr;
#endif
```

sar Data File Structure

The structure of the binary daily data file is:

```
struct paginginfo {
        unsigned int freeslots;
                cycles;
                faults;
                otherdiskios;
}
struct sa {
        struct sysinfo si;
        struct paging info pi;
        int szinode;      /* current entries of i-node table */
        int szfile;  /* current entries of file table */
        int sztext; /* current entries of text table */
        int szproc;       /* current entries of proc table */
        int mszinode;     /* size of i-node table */
        int mszfile;      /* size of file table */
        int msztext;      /* size of text table */
```

```
        int mszproc;       /* size of proc table */
long inodeovf;   /* cum. overflows of i-node table */
        long fileofv;       /* cum. overflows of file table */
        long textofv;       /* cum. overflows of text table */
        long procofv;       /* cum. overflows of proc table */
        time_t ts;  /* time stamp, seconds */
        long devio[NDEVS][4]        /* device info for up to NDEVS units */
#define IO_OPS        0              /* cum. I/O requests */
#define IO_BCNT       1       /* cum. blocks transferred */
#define IO_ACT        2              /* cum. drive busy time in ticks */
#define IO_RESP       3       /* cum. I/O resp time in ticks */
};
```

Communicating with System Users

If your system has more than one user, you may find it convenient to use the system to communicate about such things as changes to the system, work schedules, unexpected system shutdowns, or new information that affects that project you are working on. The services described in this section provide several convenient means for communication among users.

Communicating with Another User—The write Command

The **write** command sends a message to another user. Often, the **write** command is used to converse with another user (that is, each user alternately sends and receives a short message).

If you do not want to be interrupted by messages, enter `mesg n` to deny message permission. Enter `mesg y` to permit messages again.

The **write** command sends a sound signal to the display station of the person receiving the message, and then displays the following message on that display station:

Message from *username* (tty*nn*) [*date*] . . .

After successful connection with the other user's display station, **write** sends two sound signals to your display station.

When you try to send a message to a user who is not logged in, **write** displays the message: `user is not logged on`. If you send a message to a user who has refused message permission (with the **mesg** command), **write** displays the message: `got open of device /dev/console failed, permission denied`.

To Send a Message with write

1. Enter:

   `write` *username*

2. After the two sound signals, enter your message.
3. At the end of your message, press:

   END OF FILE.

After you send a **write** message, the other user can respond by sending a message back to you. You may find the following conventions for carrying out such an exchange of messages useful: When you first enter the **write** *username* command, wait for the other

user to send a message back before you send any text. End each message with a signal such as `o` (over) to alert the other person to reply. Use `oo` (over and out) when you complete the exchange.

For more information about **write**, see **write** in *AIX Operating System Commands Reference*. For information about using **write** in a Distributed Services environment, see "Using the write Command with Distributed Services" on page 11-44.

Sending a Message to all Logged-In Users—The wall Command

The **wall** (write all users) command sends a message to all logged-in users. If you realize that you will have to shut down the system unexpectedly, the **wall** command is a good way to tell everyone who logged in to bring their work to a stopping place and log out. You must have superuser authority to use the **wall** command.

To send a message with **wall**, enter `wall` *message*:

```
# wall System shutdown at 14:30
# _
```

The **wall** command sends the message, preceded by a heading, to every user logged in to the system, for example:

```
Broadcast Message from root on console . . .
System shutdown at 14:30
```

Creating a Message of the Day

To communicate with all users who will log in on a given day (not just those currently logged in), you can create a ***message of the day***. To create a message of the day, edit the **/etc/motd** file. Simply enter the text of your message into **/etc/motd** and then save the edited file. After you create a message of the day, the system cats a copy of the **/etc/motd** when the user logs in.

Creating and Reading News Items—The news Command

Use the **news** command to keep system users informed of news about the system or other topics of general interest. There are two tasks involved in using the **news** command:

- Posting news items
- Informing system users about how **news** works.

To post a news item, create a file in the **/usr/news** directory. You can either use an editor to create a file in **/usr/news** or copy a file from another directory into **/usr/news**. For example, if you create a file named `schedule` in your current directory, you can use the **cp** (copy) command to enter `schedule` as a news item:

```
cp schedule /usr/news
```

To create a news item, you must have write permission for the **/usr/news** directory. You should periodically remove all the files that contain outdated news items.

The **news** command gives you access to the items stored in **/usr/news**. To avoid reporting old news, the **news** command stores a ***currency time*** each time a user reads news items. **news** considers only the items posted after this time to be current for that user.

You can use the **news** command alone, with the name of one or more news items, or with one of three flags, depending upon the type of information you need. The following list explains the type of information returned by each version of the **news** command:

news Displays all items posted since you last read the news. To display the news items one page at a time, pipe the output of **news** to the **pg** command:

```
news | pg
```

news -a Displays all news items, regardless of the currency time. The currency time does not change.

news -n Reports the names of current news items without displaying their contents. The currency time does not change.

news -s Reports the number of current news items without displaying their names or contents. The currency time does not change.

news *item* Displays the contents of a particular *item*.

Typically, users enter the `news -n` command first. If **news -n** shows items that may be of interest, then the user can use the `news` *item* command to display the contents of the individual news items.

Sending and Reading Messages—The mail Command

The **mail** command allows you to:

- Send messages or files to specific users.
- Receive and process mail sent to you.

To Send Mail

1. Enter:

   `mail` *username*

   where *username* is a list of one or more user names.
2. Enter your message.
3. Press **END OF FILE**.

You can also use **mail** to send a file to other users with a command of the form:

`mail` *username* <*filename*

The **mail** command prefixes each message with the sender's name and the date and time of the message (its ***postmark***). **mail** then places messages in `/usr/mail/`*username* (for example, `/usr/mail/tom`. When users read their mail and then use the **s** subcommand to save the message, **mail** saves the file in the user's mailbox—that user's **$HOME/mbox** file (unless the user specifies a different file name).

To Read Mail

1. Enter:

   `mail`

2. At the ? prompt following each message, press **Enter** to read the next message.
3. (Optional) To display the previous message, enter:

   `-` (hyphen)

4. (Optional) To delete the current message and display the next message, enter:

   `d`

5. (Optional) To save the message, enter:

   `s` *filename*

   If *filename* is not specified, **mail** saves the message in **/$HOME/mbox**.
6. (Optional) To forward a message to another user, enter:

   `m` *username*

7. To leave any remaining mail in the **/usr/mail/***username* file and end the **mail** program:

   Enter `q` or press **END OF FILE**.

For information about other **mail** flags and subcommands, see **mail** in *AIX Operating System Commands Reference*.

Identifying Logged-In Users—The who Command

Use the **who** command to identify all users currently logged-in to the system. Besides showing the user name of each logged in user, **who** also reports which display station each user is using and the date and time that each user logged in. In the following example, **who** reports that there are three users logged in to the system:

```
$ who
sam       console      Apr   4  09:19
pat       tty0         Apr   4  13:31
mark      tty2         Apr   4  15:04
$ _
```

For information about other ways to use the **who** command, see **who** in *AIX Operating System Commands Reference.*

Managing Ports, Cables, and Modems

This section provides an overview of the role of ports, cables, and modems in establishing communications between a local and a remote computer.

The section includes the following information:

- Configuring ports, including types of ports, using port commands, and setting up a port
- Connecting terminals and modems, including using adapters, cables, and null modem cables
- Configuring modems, including modem connections for both call-out and call-in modems, and modem switch settings.

Two sample modem connections show the relationship between configuring modems and configuring the Basic Networking Utilities (BNU) communications facility for use with modem connections. For information about BNU, see Chapter 9, "Managing the Basic Networking Utilities."

Ports

This section provides an overview of ports and tells you how to perform the following operations:

- Use the port commands **pstart**, **penable**, **pdelay**, **pshare**, **pdisable**, and **phold** to change the port configurations at your site
- Use the **devices** command to customize port configurations to work with the communications facility used at your site.

As used in this section, a ***port*** is the logical connection between your RT and another piece of equipment such as a computer, a printer, or a terminal. A port is associated with certain programs and files.

Physically, a port is supported by an adapter in the RT. The adapter connects the computer to another piece of equipment using one of the following:

- A direct cable or a null modem cable that permanently connects an RT or a terminal to the adapter. This is called a ***hardwired connection***.
- A modem and telephone line that connect temporarily with remote systems for both local and long-distance calls
- A short-haul modem, which is a special type of hardwired connection that functions like a modem for short distances.

Types of Ports

A port is set up either for an outgoing or an incoming call. Some ports are always set to a ***call-in*** state, while others are always set to a ***call-out*** state:

Call-out State In this state, a local system initiates a connection to a remote system. In order to make an outgoing call, the port must be ***disabled*** using the **pdisable** command. Executing this command prevents another system from attempting to log in on the local system.

A call-out port is also referred to as a ***primary site***.

Call-in State In this state, a local system receives a login from a remote system. The port must be ***enabled***, using the **penable** command, to receive information from a remote system.

If the only port on a system is a call-in port, that port is referred to as a ***secondary site***.

Still other ports are set to switch functions automatically in response to a port command. These are called ***bidirectional ports***. For example, when a user on a local system issues a BNU command to send a file to a remote system, the port used to make the connection is disabled. On the other hand, a port is enabled when the local system receives a file sent by BNU from another system.

Using the Port Commands

There are six commands that you use to enable and disable ports:

pstart *port_name* Enables all ports (normal, shared, and delayed) that are enabled in the **/etc/ports** file. If you do not specify a device to enable, **pstart** reports the names of all enabled ports and tells whether they are currently enabled as normal, shared, or delayed. Usually the command is run in the form **pstart -a -i -w** from **/etc/rc** to enable all ports on a multi-user system.

penable *port_name* Enables normal ports that are enabled in the **/etc/ports** file. Normal ports are ports that are asynchronous and only allow users to log in to those ports. No outgoing use of the port is allowed while it is enabled. This command is equivalent to the statement **penable enabled=true**. If you do not specify a device, **penable** reports the names of the currently enabled normal ports.

pshare *port_name* Enables shared ports that are enabled in the **/etc/ports** file. Shared ports are bidirectional. This command is equivalent to the statement **pshare enabled=share**. If you do not specify a device, **pshare** reports the names of the currently enabled shared ports. To enable shared ports, **getty** attempts to create a lock file in **/etc/locks** that contains the ASCII process ID of the

getty process. If the port is already in use by some other process, **getty** waits until the port is available and tries again.

pdelay *port_name*
: Enables delayed ports that are enabled in the **/etc/ports** file. Delayed ports are ports that are enabled like shared ports except that the login herald is not displayed until the user types one or more characters (usually carriage returns). If the port is directly connected to a remote system or connected to an intelligent modem, the port is usually enabled as a delayed port to prevent the **getty** from talking to a **getty** on the remote side, or to the modem on a local connection, thereby consuming system resources. This statement is equivalent to **pdelay enabled=delay**. If you do not specify a device, **pdelay** reports the name of the currently enabled delayed ports.

pdisable *port_name*
: Disables the port. The local system cannot accept a remote login request.

phold *port_name*
: Specifies that an enabled port should be disabled as soon as the current user logs out.

These commands work by changing the status of the port that is specified in the **/etc/portstatus** file. When the port status is changed with one of the port commands, that change is only temporary. Once the system is shut down and then restarted, the port status reverts to the default value, which is specified with the **devices** command.

You can use the port commands to change a single port or a class of ports. Following are some examples of using the port commands.

- The following command disables all ports configured for call-ins:

  ```
  pdisable term = dialin
  ```

- This next example command enables all normal, shared, and delayed ports that are enabled in the **/etc/ports** file, reinitializes the existing **/etc/ports**, and makes the command return immediately rather than wait for **init** to confirm port status:

  ```
  pstart -a i -w
  ```

- This example enables the system attached to the **/dev/tty0** port as a shared port:

  ```
  pshare /dev/tty0
  ```

- The next example enables all 9600 baud ports as delayed:

  ```
  pdelay speed=9600
  ```

- The following command disables logins from remote computers using a specified device:

  ```
  pdisable tty2
  ```

How to Set Up a Port (devices)

To set up a port, you must perform the following actions:

- Specify the adapter port and the device to which you want to connect
- Select the device settings that create a call-out, call-in, or bidirectional port.

To start both tasks, issue the **devices** command and then select the **add** subcommand from the Device Customizing Commands menu.

You must know the following information when setting up a port:

- Device class
- Device type
- Adapter name
- Port number, if requested
- Any changes required in the default device settings.

Note: If you change the default settings at a later time, disable the device before issuing the **devices** command.

For information on devices and the **devices** command, see *Installing and Customizing the AIX Operating System.* For examples of specifying the two types of modem connections, see "BNU Modem Connections for a Call-out Port" on page 5-43 and "Configuring a Call-in Port" on page 5-46.

Connecting Terminals and Modems

This section briefly describes the adapters and cables that you need to connect terminals and modems to a system through asynchronous ports.

For more detailed information, see *User Setup Guide* and *Site Planning and Preparation Guide.*

Using Adapters

You can use one of three serial adapter cards, or one of the serial ports on the system board, for communications between a local and a remote system:

- An AT serial/parallel card (for a modem attachment)
- A multiport RS-232 card (for a modem attachment or a hardwired connection)
- A multiport RS-422 card (for a hardwired connection)
- The 6150 system board for a modem attachment or a hardwired connection.

Using Direct Cables

Consult *Site Planning and Preparation Guide* and *User Setup Guide* for descriptions and drawings of the various direct cable connections.

Using a Null Modem Cable

When two devices that both function as DTEs (data terminal equipment) are directly connected, the cable that connects them must transpose send/receive signals. A particular type of device, called a ***null modem cable***, is usually required.

The null modem cable connects a terminal directly to a computer port through a wired connector with a specific pin arrangement.

Modems

There are two types of modems: ***internal*** and ***external***.

An internal modem is an adapter that fits into the RT. An external modem is a box outside the system unit that plugs into a single-port or four-port adapter.

You must configure the modem connection for the specific modem that you use. The instructions that come with the modem provide a description of its characteristics.

BNU Modem Connections for a Call-out Port

The following examples show how to customize and configure BNU to use a call-out port with a Hayes 1200 modem. (Other modems may require different setting.) This connection enables the local system to call out to a remote system.

- For detailed information about customizing the BNU files for connecting two computers over a telephone line, using modems, see "Setting Up Remote Communications" on page 9-20.
- For information about the **devices** command, see "How to Set Up a Port (devices)" on page 5-42.

Following are the characteristics of the modem for which you want to set up a BNU entry on the local system:

Type of Modem	Hayes, 1200 baud
Name of the Calling System	zeus (local system)
Name of the Device	tty1
Destination Name	odin (remote system)

Destination Phone Number	9-1-555-123-4567 (remote system)
Destination Login ID	uucp (remote system)
Destination Password	biguucp (remote system)

The following configuration enables the local system `zeus` to call out from the local modem to the modem connected to the remote system `odin`.

Configuring a Call-out Port for a Local Modem

1. Add the following entry to the **/usr/adm/uucp/Devices** file:

   ```
   ACU  tty1  -  1200  hayes  \T
   ```

   This specifies the device that the local system `zeus` will use to communicate with the remote system `odin`.

2. Add the following entry to the file **/usr/adm/uucp/Systems**:

   ```
   odin Any ACU 1200 local4567 in:--in: uucp word: biguucp
   ```

   This entry specifies that `zeus` can call `odin` at any time on any day.

3. Add the following entry to the **/usr/adm/uucp/Dialcodes** file:

   ```
   local 9-1-555-123
   ```

   This specifies the dialing code prefix used in the **Systems** file.

4. Add the following lines to the **/usr/adm/uucp/Permissions** file:

   ```
   LOGNAME=uucp VALIDATE=odin REQUEST=yes SENDFILES=yes READ=/ WRITE=/
   MACHINE=zeus:odin REQUEST=yes COMMANDS=ALL READ=/ WRITE=/
   ```

   This specifies the access that `odin` has to `zeus`.

5. Add device **tty1** by entering the information in the next quick reference box.

For information about configuring the BNU files mentioned above, see the following:

- For information about the **Devices** file, see "Setting Up Devices" on page 9-22.
- For information about the **Dialers** file, see "Setting Up Dialers" on page 9-30.
- For information about the **Systems** file, see "Customizing the Systems File" on page 9-33.

- For information about the **Dialcodes** file, see "Setting Up Dialing Codes" on page 9-42.
- For information about the **Permissions** file, see "Customizing the Permissions File" on page 9-43.

Adding a Device for a Call-out Port

1. Enter the **devices** command:

   ```
   devices
   ```

2. When the Device Customizing Menu appears, enter the **add** option, the device class, and the device type:

   ```
   a ttydev tty
   ```

3. In response to the prompts, select the first RS-232C, 4-port asynchronous adapter and the first port and press the **Enter** key:

   ```
   rs232c1 1
   ```

4. When prompted about changes in the pre-determined information in the system configuration, type **yes** to display and change the current settings.
5. Now set the device options as listed in the next quick reference box.

Setting Device Options for a Call-out Port

1. Set the options for the device specified in the above example as follows:

   ```
   bpc   8 (data sent 8 bits per second)
   pt    none  (no parity)
   rts   1200 (receive and transmit speed 1200 bps)
   sns   use default, or false
   aa    false (no automatic answer for call-out)
   dvam  1 (device attached by modem)
   nosb  1 (data has one stop bit)
   om    full (mode of operation is duplex)
   pro   dtr (protocol is dtr--data terminal ready)
   ixp   true (include Xon/Xoff protocol)
   ```

2. Use the default for all other options.
3. Press the **Enter** key twice.

Note: If you change a port configuration rather than create a new port, enter **penable** and then **pdisable** to put the new settings for the call-out port into effect.

Configuring a Call-in Port

This example shows how to customize and configure BNU to use a call-in port with a Hayes 1200 modem and an RS-232 card. (Other devices may require different setting.) The call-in port is on the local computer.

- For detailed information about customizing the BNU files for connecting two computers over a telephone line, using modems, see "Setting Up Remote Communications" on page 9-20.
- For information about the **devices** command, see "How to Set Up a Port (devices)" on page 5-42.

Following are the characteristics of the modem for which you want to set up a BNU entry on the remote system:

Type of Modem	Hayes, 1200 baud
Name of the Calling System	odin (remote system)
Name of the Device	tty1

Destination Name	zeus (local system)
Destination Phone Number	9-1-555-765-4321 (local system)
Destination Login ID	uucp (local system)
Destination Password	alluucp (local system)

Configuring a Call-in Port for a Remote Modem

1. Add the following entry to the file **/usr/adm/uucp/Systems**:

   ```
   zeus Any ACU 1200 local4321 in:--in: uucp word: alluucp
   ```

   This entry specifies that `odin` can call `zeus` at any time on any day.

2. Add the following lines to the **/usr/adm/uucp/Permissions** file:

   ```
   LOGNAME=uucp VALIDATE=zeus REQUEST=yes SENDFILES=yes READ=/ WRITE=/

   MACHINE=odin:zeus REQUEST=yes COMMANDS=ALL READ=/ WRITE=/
   ```

3. Add device **tty1** by entering the information in the next quick reference box.

For information about configuring the BNU files mentioned above, see the following:

- For information about the **Devices** file, see "Setting Up Devices" on page 9-22.
- For information about the **Dialers** file, see "Setting Up Dialers" on page 9-30.
- For information about the **Systems** file, see "Customizing the Systems File" on page 9-33.
- For information about the **Dialcodes** file, see "Setting Up Dialing Codes" on page 9-42.
- For information about the **Permissions** file, see "Customizing the Permissions File" on page 9-43.

Adding a Device for a Call-in Port

1. Enter the **devices** command:

   ```
   devices
   ```

2. When the Device Customizing Menu appears, enter the **add** option, the device class, and the device type:

   ```
   a ttydev tty
   ```

3. In response to the prompts, select the first RS-232C, 4-port asynchronous adapter and the first port and press the **Enter** key:

   ```
   rs232c1 1
   ```

4. When prompted about changes in the pre-determined information in the system configuration, type **yes** to display and change the current settings.
5. Now set the device options as listed in the next quick reference box.

Setting Device Options for a Call-in Port

1. Set the following:

   ```
   aa true (automatic answer for call-in)
   ```

2. All other settings are the same as for the call-out port.
3. Press the **Enter** key twice.
4. Enter the following to complete the configuration for the call-in port:

   ```
   penable tty1
   ```

Note: The steps described for configuring a call-in port are an example of the general procedure you follow to perform this task. You may need to set other parameters to match those of the remote system, such as transmission speed, parity, and bit length/character.

The procedure for adding a device for a bidirectional port is exactly the same as that for adding a device for a call-in port, as described in the quick reference box above.

The procedure for setting the device options for a bidirectional port is almost the same as for a call-in port. However, instead of setting `ae true`, set the following:

```
ae delay
```

Switch Settings for a Hayes Smartmodem 1200

In the settings that follow, Up means "open", and Down means "closed". The numbers represent the modem switches—that is, 1 represents switch number 1, 2 represents switch number 2, and so on.

1:	**Up**	DTR yes
2:	**Up**	Word result codes displayed
3:	**Down**	Result codes sent
4:	**Down**	Characters echoed in command state
5:	**Up/Down**	Autoanswer enabled/disabled This setting should be Up for a *call-in port*, and Down for a *call-out port.*
6:	**Down**	RS-232C Carrier Detect on (always)
7:	**Up/Down**	Single line/multiline This setting should be Up for a *standard RJ11 phone jack*, and Down for a *multiline phone system.*
8:	**Down**	Command recognition enabled
9:	**Up**	Compatible with Bell 103/212A modem
10:	**Up**	Modem reset when turned on

Configure the associated terminal port with modem control.

Chapter 6. Managing System Security

CONTENTS

wt: managing the AIX
op. system.

About This Chapter

This chapter discusses the role that an administrator of the system plays in maintaining proper security. Certain functions of the AIX Operating System strengthen security. In a secure system, access to files and directories is more highly regulated and restricted than in non-secure systems. This chapter gives you the background you need to manage system security and helps you understand the risks of certain policies and functions. With this background and understanding, you can establish a proper balance between sharing and protecting resources.

Security Administration

Proper system administration is vital to maintaining the security of the information resources of a computer system. AIX security is based on establishing and maintaining proper access control and accountability policies. It is an administrator's responsibility to configure the following aspects of security:

- Access control
- Identification and Authentication restrictions
- Trusted Computing Base (TCB)
- User auditing.

Once these features have been configured, a person who administers the system will be able to:

- Control access to information resources, hard-copy material, and terminals
- Configure passwords
- Install and manage the TCB and software packages
- Audit user actions.

Aspects of Computer Security

The primary goal of security is the detection and prevention of security violations on a system. Computer security includes the following fundamental aspects:

Access Control
This aspect of security addresses the privacy, integrity and availability of information on your system.

Identification and Authentication
This procedure is concerned with how users are identified and how their actions are traced by the system.

Trusted Computing Base
The TCB is the part of the system that is responsible for enforcing the information security policies of the system.

Auditing
Auditing is the recording and analysis of actions that take place on the system.

Access Control

The AIX Operating System access control involves what information resources are protected and who can be granted access to those resources. The AIX Operating System allows for need-to-know or discretionary security. The owner of an information resource can grant other users read or write access rights for that resource. A user who is granted access rights to a resource may transfer those rights to other users. This security allows for user-controlled information flow in the system; the owner of an information resource defines the access permissions to the object.

Users have user-based access only to the objects they own. Typically, users will receive either the group permissions or the default permissions for a resource. The major task in administering access control is to define the group memberships of users because these memberships determine the users' access rights to the files they do not own.

Access control in the AIX Operating System involves managing protected resources through **setuid** and **setgid** programs, and hard-copy labelling.

Managing Protected Resources

The AIX Operating System supports several types of information resources, or objects. These objects provide a way for user processes to store or communicate information.

The most important types of objects are:

- Files and directories (used for information storage)
- Named pipes, message queues, shared memory segments and semaphores (used for information transfer between processes).

Each object has an associated owner, group, and mode. The mode is used to define access permissions for the owner, the group, and for all other users.

Access Authorization

Managing access rights is the responsibility of the owner of the information resource. Resources are protected by the permission bits, which are included in the mode of the object. The permission bits define the access permissions granted to the **owner** of the object, the **group** of the object, and for the default class **others**. AIX supports three different modes of access (usually read, write, and execute) that can be granted separately.

When a user logs into an account (using **login** or **su**), the user ID and group IDs assigned to that account are associated with the user's processes. These IDs determine the access rights of the process.

When a process requests access to an information resource, the following permissions criteria apply:

- If the process's user ID matches that of the owner of the resource, then the process receives the access permissions of the owner.
- If one of the process's group IDs matches that of the group of the resource, then the process receives the access permissions of the group.
- Otherwise, the process receives the default access permissions.

If the requested type of access is included in the access permissions for the process, access is granted; otherwise it is denied.

Note: A process with a user ID of 0 is known as a superuser process. These processes are generally allowed all access permissions. But if a superuser process requests execute permission for a program, access is granted only if execute permission is granted to at least one user.

The AIX discretionary access control mechanism allows for effective control of access to information resources and provides for separate protection of the confidentiality and integrity of the information. Owner-controlled access control mechanisms, like this one, however, are only as effective as users make them. All users must understand how access permissions are granted and denied and how these are set. For more information on setting file and directory permissions and how permission bits work, see *Using the AIX Operating System.*

Using setuid and setgid Programs

The permission bits mechanism allows effective access control for most resources in most situations. But for more precise access control, the AIX Operating System provides **setuid** and **setgid** programs.

Normally, programs execute with the user and group access rights of the user who invoked them. Program owners may associate their access rights with a program by making it a **setuid** or **setgid** program; that is, a program with the **setuid** or **setgid** bits set in its permissions field. When that program is executed by a process, the process acquires the access rights of the owner of the program. A **setuid** program executes with the access rights of its owner, while a **setgid** program will have the access rights of its group. Note that both bits may be set.

Although the process is assigned the additional access rights, these rights are controlled by the program bearing the rights. Thus, **setuid** and **setgid** programs allow for user-programmed access controls in which access rights are granted indirectly. The program acts as a trusted subsystem, guarding the user's access rights.

Although these programs can be used with great effectiveness, they can be a security risk if they are not designed carefully. In particular, the program must never return control to the user while it still has the access rights of its owner. This control would allow that user to make unrestricted use of those rights.

Administrative Access Rights

AIX provides privileged access rights for system administration. System privilege is based on user and group IDs. Users with effective user or group IDs of 0 are recognized as privileged.

Processes with effective user IDs of 0 are known as superuser processes and can:

- Read or write any object
- Call any system function
- Perform certain subsystem control operations by executing setuid-root programs.

You can manage the system using two types of privilege: **su** command privilege and setuid-root program privilege. The **su** command allows all programs you invoke to function as superuser processes. Although this provides a flexible way to manage the system, it is not very secure.

You can also make a program into a setuid-root program. A setuid-root program is a superuser-owned program with the **setuid** bit set. A setuid-root program provides administrative functions that a non-superuser can perform without compromising security; the privilege is encapsulated in the program rather than granted directly to the user.

Although it is difficult to encapsulate all necessary administrative functions in setuid-root programs, it is a secure method of managing a system.

For information about how to use the **su** command, see **su** in *AIX Operating System Commands Reference.*

Hard-copy Access Labelling

To control access to printed output, the printed material must be labelled with the access control information for the particular document. AIX allows access control information to be printed on the following pages of output:

Header	Precedes first page of output
Trailer	Follows last page of output
Top	Top of each page
Bottom	Bottom of each page.

As the the person in charge of administering the system, you configure the security labelling of hard-copy output. Each of the above label fields is an attribute in the **printer** stanza of the **/etc/security/config** file. When the **print** command is invoked, the specified labels are printed on the hard copy.

Each label may be specified as:

- Nothing - the default
- Quoted strings

- Access control characteristics, which are:
  - Lists of users who can read data
  - Lists of groups who can read data
  - An indication of whether the public can read data.

  Note: In the case of redirected program output (where there are no explicit access control attributes), the label indicates that only the user who generated the output is allowed to read it.

For example, the following could be a printer stanza in the **/etc/security/config** file:

```
printer:
        enabled = true
        header = "COMPANY CONFIDENTIAL"
        bottom = "CLASSIFIED INFORMATION:%DAC"
```

The escape sequence %DAC is expanded into the list of access control characteristics of the document. This list will be printed on the hard copy in the specified location.

See *AIX Operating System Technical Reference* for a complete description of the **/etc/security/config** file.

Because printing labels on pages formatted with pagination or graphics can cause problems, a user can specify the **-nl** option of the **print** command. This option overrides the top-of-page and bottom-of-page label specifications. When a user requests this option and the administrator has specified top-of-page or bottom-of-page labels, **print** logs an event on the audit trail.

Trusted Communication Path

The AIX Operating System trusted communication path allows for secure communication between users and the trusted computing base. Users invoke the trusted communication path by pressing the secure attention key (SAK). This allows only trusted processes to access the user's terminal. A trusted communication path is used whenever the user must enter data that must not be compromised (a password, for example). A trusted communication path may also be used by the person who administers the system to establish a secure environment for administration.

Understanding the Trusted Communication Path

The AIX trusted communication path is based on:

- A trusted command interpreter (**tsh**) which will execute only commands marked as trusted
- Restricting access to a terminal to trusted programs
- A reserved key sequence (the secure attention key, or SAK) which allows the user to request a trusted communication path.

A trusted communication path exists when only trusted programs have been run since old accesses to the terminal were revoked.

After SAK has been pressed, the **init** process runs either the **getty** or **shell** program to:

- Change the owner and mode of the terminal so that only processes run by **root** can **open** the terminal
- Issue a **frevoke** system call to invalidate all previous **open**s of the terminal.

This assures that only the current **getty** or **shell** program has access to the terminal. See *AIX Operating System Commands Reference* for more information on **getty** and **shell**.

Trusted Command Interpreter

The **init** program runs the **shell** program in response to a SAK only if another user was logged on to this terminal. This command establishes the terminal modes and runs the AIX trusted shell, **tsh**.

The **tsh** command provides a subset of the functions of the normal AIX shell (Bourne shell). The trusted shell will execute only trusted programs. A built-in command, **shell** allows the user to drop the trusted communication path and execute the user's login shell. See *AIX Operating System Commands Reference* for more information on **tsh**.

Restricting Access to a Terminal

As mentioned above, the **getty** and **shell** programs change the owner and mode of a terminal to prevent untrusted programs from accessing the terminal. If the terminal can be accessed by another name, this protection is ineffectual unless the other name is similarly protected. The AIX Operating System provides a way to configure exclusive terminal access.

The synonym Attribute

The **synonym** attribute in **/etc/ports** indicates to the trusted programs that a terminal can be accessed by another name. For example, **/dev/console** can also be referenced through **/dev/hft**. Similarly, most asynchronous terminals can be accessed with or without modem control as **/dev/ttyXX** or **/dev/lttyXX**, respectively.

The shell attribute

Establishing a trusted communication path on the console for **login** involves revoking access to all virtual terminals. This means when a user logs off the console, all processes running on other virtual terminals are likely to be killed. To lessen this possibility, the **shell** attribute in **/etc/ports** may be used to specify a program which is to be run as a manager on this terminal. In the case of **/dev/console**, the **/bin/actman** program should be run as a manager. This program provides a warning to users if they attempt to log off the console while other virtual terminals are in use. See *AIX Operating System Commands Reference* for more information on **actman**.

Using the Trusted Communication Path

A trusted communication path is established by pressing a reserved key sequence **(Ctrl-X, Ctrl-R)**, called the secure attention key (SAK)

A trusted communication path should be established:

- When you log in to the system. After you press the SAK:
  - If a new login screen scrolls up, you have a secure path.
  - If the trusted shell prompt appears, the initial login screen was an unauthorized program that may have been trying to steal your password. You should find out who is currently using this terminal (using the **who** command) and then log off.
- When you want to be certain that the command you enter will result in a trusted program running. For example,
  - You should run as root only after establishing a trusted communication path. This ensures that no untrusted programs will be run with superuser authority.
  - You should run the **su**, **passwd**, and **newgrp** programs only after establishing a trusted communication path.

Warning: Use caution when using SAK. It kills all processes which attempt to access the terminal.

Configuring the Secure Attention Key

Each terminal may be independently configured as to whether pressing SAK at that terminal will create a trusted communication path. This is specified by the SAK attribute in **/etc/ports**. If the value of this attribute is **on**, recognition of the SAK is enabled.

The line:

```
sak = on
```

should be added to the **default** stanza of **/etc/ports**. This enables SAK recognition on all terminals. If a port is to be used for communications, (for example, by **uucp**), the specific port used should have the line:

```
sak = off
```

This line disables the SAK.

Identification and Authentication

Identification and Authentication is the establishment of the user's identity. Each user is required to log in to the system. The user supplies the user name of an account and a password, if the account has one. (In a secure system, all accounts should either have passwords or be invalidated.) If the password is correct, then the user is logged into that account; the user acquires the access rights and privileges of the account. See "Configuring Password Restrictions" on page 6-12, "The /etc/passwd File" on page 2-27, and "The /etc/security/passwd File" on page 2-29 for additional information about password files.

Understanding Identification and Authentication

AIX provides a password mechanism for user authentication. All user accounts designed for normal usage should have passwords. An individual who knows the password for an account may log in to that account and gain its access rights and privileges. Groups also may have passwords, and a user who is not a defined member of the group may still acquire its access rights by knowing its password.

Because the password is the only protection for each account, it is important that users select and guard their passwords carefully. Many attempts to break into a system start with attempts to guess passwords. AIX provides significant password protection by storing user and group passwords separately from other user and group information. The encrypted passwords and other security relevant data for users and groups are stored in the files **/etc/security/passwd** and **/etc/security/group**, respectively. These files should be accessible only by the superuser. With this restricted access to the encrypted passwords, an attacker cannot decipher the password with a program which simply cycles through all possible or likely passwords.

It is still possible to guess passwords by repeatedly attempting to login to an account. If the password is trivial or is infrequently changed, such attacks may easily succeed. As a result, it is also important to educate users on the importance of choosing passwords which cannot easily be guessed.

Some of the obvious trivial passwords are passwords that:

- Can be found in a dictionary
- Include the names of users or one of their close relatives
- Include the current month or year
- Are based on some attribute of the user.

Configuring Password Restrictions

Proper password management can only be accomplished through user education. But to provide some additional security, AIX provides configurable password restrictions. These allow the administrator to constrain the passwords chosen by users, and to force passwords to be changed regularly. These restrictions are recorded in the **password** stanza of the **/etc/security/config** attribute file, and are enforced whenever a new password is defined for a user or group. Note that all restrictions are per system, not per user; if proper password security is to be maintained, then all passwords should be similarly protected.

The restrictions that can be applied are:

minage The minimum number of weeks that must pass before a password can be changed.

maxage The maximum number of weeks that can pass before a password must be changed.

minalpha The minimum number of alphabetic characters the new password must contain.

minother The minimum number of non-alphabetic characters the new password must contain. (Other characters are any ASCII printable characters which are nonalphabetic and are not National Language code points).

Note: The minimum length of a password on the system is **minalpha** + **minother**. The maximum length of a password is 8 characters.

maxrepeat The maximum number of times a character can appear in the new password.

mindiff The minimum number of characters in the new password that must be different from the characters in the old password.

The following table gives recommended, default, and maximum values for each of these restrictions. The default values are applied if the **password** stanza is missing from the **/etc/security/config** file, or if a particular attribute is not defined. Secure environments should use the recommended values, and even systems used only as personal work stations should require a password of some sort, especially if they are connected to any network.

Restriction Values	Advised Values	Default Values	Maximum Values
minage	1	0	52
maxage	8	0	52
minalpha	5	0	8
minother	2	0	8*
mindiff	3	0	8
maxrepeat	1	8	8

* The maximum value of minother is (8) - (minalpha).

Restrictions should be set so that users select passwords that are hard-to-guess, not hard-to-remember. Passwords that are hard to remember are often written down somewhere, which compromises system security.

Login User IDs

AIX also identifies users by their ***login user ID***. The login user ID allows the system to trace all user actions to their source. After a user logs in to the system, but before running the initial user program, the system sets the login ID of the process to the user ID found in the user data base. All subsequent processes during the login session are tagged with this ID. These tags provide a trail of all activities performed by the login user ID.

The user can reset the ***real user ID, real group ID***, and ***concurrent group*** set during the session, but cannot change the login user ID. All audit events recorded for this user are labeled with this ID and should be examined when you generate audit records.

Trusted Computing Base

The trusted computing base (TCB) is the part of the system that is responsible for enforcing the information security policies of the system. All of the computer's hardware is included in the TCB, but a person administering the system should be concerned primarily with the software components of the TCB.

The TCB software in the system consists of:

- The kernel of the operating system
- The configuration files that control system operation
- Any program that is run with the privilege or access rights to alter the kernel or the configuration files.

Most system files are accessible only by the superuser; however, some may also be accessed by members of the system group. Only the superuser can alter the operating system kernel. The TCB contains the following, which are trusted programs:

- All setuid root programs
- All setgid system programs
- Any program that is run by the superuser or by a member of the system group.

In the AIX Operating System, the person who administers the system can mark trusted files as part of the trusted computing base, so they can be clearly distinguished. See the **chtcb** command in *AIX Operating System Commands Reference*.

The person who administers the system must be careful to add only software that can be fully trusted to the TCB. You may choose to trust software for several reasons:

- You have fully tested the program
- You have examined the program's code
- The program was obtained from a source that you trust to have done one of the above.

It is up to the person who administers the system to determine how much trust can be given to a particular program. The administrator must also consider the value of the information resources on the system in deciding how must trust is required for a program to be installed with privilege.

Checking the Trusted Computing Base

The **sysck** program audits the security state of the operating system. The security of the AIX operating system is jeopardized when the TCB files are not properly protected or when configuration files have unsafe values. The **sysck** program audits this information by reading the **/etc/security/sysck.cfg** file. This file includes a description of all TCB files, configuration files, and trusted programs.

Using the sysck Program

The **sysck** program is normally used to:

- Assure the proper installation of security-relevant files
- Assure that the file system tree contains no files which clearly violate the system security
- Update, add, or delete trusted files.

Using sysck on Trusted Files

You should run the **sysck** program to check the installation of trusted files at system initialization. To perform this automatically, add the following command to the **/etc/rc** file:

```
sysck -p all
```

This will cause the **sysck** program to check the installation of each file described by **/etc/security/sysck.cfg**.

Note: One of the functions of the **secure** command is to add the **sysck -p all** command to **/etc/rc**. Add the **sysck -p all** command to **/etc/rc** only if you have not executed the **secure** command.

Using sysck on the File System

You should run the **sysck** program to check the file system any time you suspect the integrity of the system may have been compromised. This is done by the command:

```
sysck tree
```

When **sysck** is invoked with the **tree** parameter, all files described by **/etc/security/sysck.cfg** are checked for correct installation. In addition, the following checks are performed on all other files in the file system. If the **sysck** program discovers any files that are potential threats to system security, you can alter the suspected file or add a description of the file to the **/etc/security/sysck.cfg** file.

- If the file owner is **root** and the file has the **setuid-on-exec** bit set, the **setuid-on-exec** bit is cleared.
- If the file group is **system**, the file is executable, and the file has the **setgid-on-exec** bit set, the **setgid-on-exec** bit is cleared.
- If the file has the **tcb** attribute set, this attribute is cleared.
- If the file is a device (character or block special file), the access mode of the file is set to 0.
- If the file is an additional link to a path name described in **/etc/security/sysck.cfg**, the link is removed.
- If the file is an additional symbolic link to a path name described in **/etc/security/sysck.cfg**, the symbolic link is removed.

Using sysck to Add a Trusted Program

To add a specific program to the **/etc/security/sysck.cfg** file, use the command:

```
sysck -a pathname [attribute=value]
```

Only attributes whose values can or should not be deduced from the current state of the file need be specified on the command line. A table of all attribute names appears in "Configuring the sysck Program" on page 6-17.

For example, the following command will register a new setuid-root program called **/usr/bin/setgroups** which has a link named **/usr/bin/getgroups**:

```
sysck -a /usr/bin/setgroups links=/usr/bin/getgroups
```

After installing a licensed program (LPP), you may not know which new files should be registered in the **/etc/security/sysck.cfg** file. These may be found and added with the command:

```
sysck tree
```

This command will display the name of any file which should be registered in the **/etc/security/sysck.cfg** file. With this command you can alter the **/etc/security/sysck.cfg** file and add program descriptions to it.

Using sysck To Delete a Trusted Program

If you remove a file described in the **/etc/security/sysck.cfg** file, you should also remove the description of this file. For example, if you have delete the **/etc/cvid** program, the following command:

```
sysck all
```

will print a message of the form:

```
** ERROR:    "/etc/cvid" - does not exist
```

The description of this program can be removed with the command:

```
sysck -d /etc/cvid
```

Configuring the sysck Program

The **sysck** program reads the **/etc/security/sysck.cfg** file to determine which files to check. Each trusted program on the system should be described by a stanza in the **/etc/security/sysck.cfg** file.

Each stanza has the following attributes:

Attribute	Description
class	The name of a group of files. This attribute allows several files (with the same **class** name) to be checked by specifying a single argument to the **sysck** command.
owner	The user ID or user name of the file's owner. If this does not match the file owner, **sysck** sets the owner ID of the file to this value.
group	The group ID or group name of the file's group. If this does not match the file owner, **sysck** sets the owner ID of the file to this value.
mode	The octal or symbolic value of the file's mode. This attribute value may be either a number (for example, mode = 40750) or symbolic (for example, mode = drwxr-x---). If this does not match the file mode, **sysck** sets the mode of the file to this value.
tcb	A true or false indicator of whether the file is part of the trusted computing base. If this does not match the file **tcb** attribute, **sysck** changes this attribute on the file.
links	A comma-separated list of path names linked to this file. If any path name in this list is not linked to the file, **sysck** creates the link. If invoked without the **tree** argument, **sysck** prints a message that there are extra links, but will not determine their names. If invoked with the **tree** argument, **sysck** also prints any additional path names linked to this file.
symlinks	A comma-separated list of path names symbolically linked to this file. If any path name in this list is not a symbolic link to the file, **sysck** creates the symbolic link. If invoked with the **tree** argument, **sysck** also prints any additional path names which are symbolic links to this file.

Attribute	Description
program	An associated program that can check additional status. The **sysck** program will run this specified program with the same flags as were supplied to **sysck**. See "sysck Checking Programs." **Note:** If arguments are passed, they must be enclosed in double quotes because spaces are delimiters.

If a stanza in the **/etc/security/sysck.cfg** file does not specify an attribute, the corresponding check is not performed.

sysck Checking Programs

An important aspect of the **sysck** program is the **program** attribute, found in the **sysck.cfg** file. The **program** attribute lists an associated program that can check additional status. This attribute allows for more thorough and flexible checking than other attributes provide.

You can use these checking programs to check the integrity and consistency of a file's contents and its relationship with other files. Checking programs need not be bound to a particular file.

For example, assume you wrote a program, **/etc/profiles**, which verifies that each user's **.profile** file is writable only by that user. The program:

- Is owned by **root**
- Has a group of **system**
- Has a mode of 0750
- Is tagged as part of the TCB.

You can add your program, **/etc/profiles**, to the system security checker by entering the following:

```
sysck  -a  /etc/profiles "program=/etc/profiles" "class=profiles"
```

This command creates the following entry in the data base:

```
/etc/profiles:
    class = profiles
    owner = root
    group = system
    mode = -rwxr-x---
    tcb = true
    program = "/etc/profiles"
```

The command:

```
sysck profiles
```

will verify the installation of the **/bin/profiles** program and run the program.

The are several requirements for **sysck** checking programs:

- The **sysck** checking programs must accept the **-p**, **-n**, and **-y** flags and handle these similarly to **sysck**.
- The **sysck** checking programs must return 0 to indicate that no errors were found and write all error messages to standard error.
- It is important to note that these programs are invoked with an effective user ID of 0; therefore they are fully privileged. They should be written *and* inspected as setuid-root programs.

AIX Checking Programs

The AIX Operating System supplies the following AIX **sysck** checking programs:

pwdchk Checks the **/etc/passwd** and **/etc/security/passwd** files for internal and mutual consistency. This program is invoked whenever **sysck** is invoked to check the **/etc/passwd** file. The checks performed are described in the *AIX Operating System Commands Reference* with the **sysck** command.

grpchk Checks the **/etc/group** and **/etc/security/group** files for internal and mutual consistency. This program is invoked whenever **sysck** is invoked to check the **/etc/group** file. The checks performed are described in the *AIX Operating System Commands Reference* under the **sysck** command.

Secure System Installation and Update

You must enable security features explicitly by configuration. See the **secure** command in *AIX Operating System Commands Reference.* Most security features are enabled by attributes specified in the security configuration attribute file, **/etc/security/config**. For more information on the **/etc/security/config** file, see *AIX Operating System Technical Reference.* See following sections for information on configuring security features:

"Hard-copy Access Labelling" on page 6-7
"Configuring Password Restrictions" on page 6-12
"Configuring the sysck Program" on page 6-17
"Auditing" on page 6-21.

Initial System Configuration

A shell script, **/etc/secure** is shipped with the AIX Operating System. This program may be run after system installation to increase the trustworthiness of your system.

Warning: Do not run the **secure** script until you have defined at least one user account on your system. The **secure** command prevents logging on to the system as root.

Before running this program, you should understand the system security description presented in this chapter. Careful programming or system configuration can not replace administrator and user awareness.

The **secure** program establishes a reasonably secure initial system configuration; specifically, this program:

- Defines **/dev/hft** as a synonym for **/dev/console** (in **/etc/ports**).
- Defines **/bin/actman** as the shell for **/dev/console** (in **/etc/ports**).
- Enables **SAK** on all terminals (in **/etc/ports**).
- Disallows login to accounts with user ID 0 (in **/etc/security/passwd**).
- Sets the root shell to the trusted shell (in **/etc/passwd**).
- Truncates **/etc/autolog**.
- Builds the **/etc/security/sysck.cfg** file.
- Adds a line to **/etc/rc** to invoke **sysck** at system initialization.

The **secure** program does *not*:

- Configure any of the AIX auditing system
- Configure any hard-copy labelling.

Installing Licensed Programs

Use the **installp** and **updatep** commands to install and update licensed programs on the system. See *AIX Operating System Commands Reference* for more information.

On a secure system, the administrator may wish to monitor the installation of software not known to be trusted. This may be done with **watch** command.

The **watch** command allows you to monitor the actions of an untrustworthy program. This command:

- Defines a new audit class consisting of events considered to be threats to the trusted computing base (TCB)
- Opens an audit stream (a channel of the multiplexed **/dev/audit** device) on which these audit events are to be reported, and displays these threatening events on its standard output
- Runs the installation with this audit class enabled.

For more information on the **watch** command, see the *AIX Operating System Commands Reference.*

Auditing

Auditing is the recording and analysis of security-relevant events caused by the processes run by each user of the system. An audit event is an action (such as a command or an access) taken on the system, which can be recorded by the system. The administrator of the system can choose which audit events are recorded for each user.

When a user's process takes some auditable action and that action is being recorded for the user, a record is appended to the system's audit trail. The record will contain the name of the event, the responsible user and process IDs, a success or failure indication, and other event-specific information. These records can later be analyzed and read to determine the extent and nature of any security violations that have occurred, as well as to detect and prevent potential violations.

An audit trail is generated in three phases:

1. Security-relevant occurrences (termed audit events) are detected, either by the kernel or by a trusted process.
2. The system determines if the event should be recorded.
3. The information is collected in one or more audit trails.

Subsequently, these trails may be analyzed for security considerations.

Using the Auditing Subsystem

To use auditing, you must first properly configure event selection and information collection. Next, the auditing subsystem must also be enabled, usually during system initialization. Last, the audit trail must be periodically analyzed so that security problems are quickly detected and corrected.

Before you can configure the auditing subsystem, you must know about the audit system files and the audit commands.

Audit Files

Audit system configuration is controlled by the following files:

/etc/security/config
: The **/etc/security/config** attribute file contains two stanzas for auditing:

  audit This stanza specifies whether the system provides records through the Bin collection interface. See "Information Collection" on page 6-26.

  auditclasses
  This stanza defines the **audit classes** for the system.

/etc/security/audit/events
: This file contains a list of all of the audit events detected in the system. Normally, this file is not altered by the administrator of the system; rather, events are added during system and licensed program installation. This attribute file contains a single stanza, called **auditpr**. Each event is an attribute in this stanza. The attribute value defines how the event can be formatted for viewing or printing.

/etc/security/passwd
: This file contains the security attributes for each user account. It is an attribute file which contains a stanza for each user. The attribute **auditclasses** is used to record the audit classes that are assigned to a particular user.

/dev/audit
: This is a device special file through which Stream collection audit records can be read.

/etc/security/audit/cmds
: This file contains a list of audit backends, which are shell command lines that contain one or more audit commands used to process audit bins. The audit bin manager reads this file and runs each backend command on each filled bin.

/etc/rc This file contains a list of shell commands that are run at system initialization. The administrator of the system can add commands to start system auditing and to specify Stream collection.

See *AIX Operating System Technical Reference* for more information about these files.

Audit Commands

The following commands are used for auditing.

auditstream The **auditstream** program provides a convenient interface to the Stream collection interface. It opens the audit device, **/dev/audit**, and reads records for the events in one or more **audit classes**. These records are written on its standard output, which is usually a pipe to another backend program.

auditapp The **auditapp** program will read a bin of audit records and append it to a file being used as the current audit trail.

auditselect The **auditselect** program reads either bins or a stream of audit records and selects records according to supplied criteria. This command allows the administrator to reduce the trail to examine it for certain types of behavior.

auditpr The **auditpr** program reads either bins or a stream of audit records and formats them for viewing or printing.

For more information about these commands, see *AIX Operating System Commands Reference.*

Event Detection

Audit events, as noted, are system actions which are detected by the operating system kernel or by a trusted process. Types of audit events include:

- Process creation
- File creation
- File update
- User login
- Failed login
- Changing the permission bits of a file

The AIX Operating System and some trusted programs detect events of these types and record these events through the audit system.

Event Selection

The events to be detected and recorded can be selected per user, using audit classes. The administrator of the system defines the audit classes (sets of audit events) for the system, and assigns one or more of these classes to each user. When users log in to the system, their initial processes are tagged with their audit classes. This process attribute is inherited by all subsequent processes.

An audit record will be generated for an audit event only if:

- System auditing is enabled
- The event is defined as part of an audit class for the system
- The process or user generating the event is tagged with such a class.

If the event is to be recorded, the audit system will construct an audit record, consisting of a header and tail. The header contains information common to each event, such as the name of the event, the event return status, the time, the process ID, and the process user IDs. The tail contains information specific to the event.

Example Audit Class Configuration

A list of the system audit events appears in the **/etc/security/audit/events** file. Audit classes need include only a very small subset of this file.

In **/etc/security/config**, the following stanza could occur:

```
auditclasses:
        general= TCBmod,TCBleak,su,newgrp,passwd,at
        export=  import,export,backup,restore,cvid,print
        system=  user,group,sysck,installp,updatep
        config=  mvmd,shutdown,reboot,rebuild,errstop,port
        root=    audit,auditbin,auditevent,auditproc
        init=    login,logout,crontab
```

These audit classes can be described as follows:

general The AIX access controls should prevent most users from altering the system state. However, no system is foolproof or error-free. When generated for a normal user, the **TCBmod** event indicates a trusted program is not doing its own auditing, or the user has managed to circumvent the AIX access control mechanisms. Similarly, the **TCBleak** event indicates a user has managed to deceive a trusted program into running untrusted software.

In addition to these events, user authentication changes (by **su**, **newgrp**, or **passwd**) and the daemon request, **at**, are selected to be recorded.

export An installation may also want to keep track of the flow of information out of (or into) the system. The **export** audit class contains the system events generated when a user writes an archive to diskette, or prints a file.

system Users in the **system** group (group ID 0) can perform administrative actions. These include:

- Modifying accounts by the **adduser** program. The **user** and **group** events are generated by the **adduser** program.
- Installing programs. The **installp**, **updatep** and **sysck** records are generated by the programs of the same names.

This audit class lists administrative actions to be auditied

config Members of the **system** group may also update the system configuration. This class is defined to record their actions.

root Once a user has become **root** (by **su**), there are no constraints on the user's actions. Such a user generally must be trusted. An installation may want to audit all actions of superusers, but it is more reasonable just to record when **root** attempts to circumvent the audit system. This class contains events concerning the audit system.

init The immediate descendants of **init** (for example, **getty**, **login**, **cron**) will generate audit records for all events in all classes defined in **/etc/security/config**. These descendents should be trusted programs that generate specific audit records. The **init** program recognizes the **logout** event, **loginx** recognizes **login**, and **cron** generates the **crontab** event. This audit class contains events generated by **init** that need to be recorded.

An example user audit configuration can be created using the above audit class definitions and three users:

Dave An administrator of the system (a member of system group)

Mary A full-time employee who updates information on the system

Bill A part-time employee who is expected to use the system only to examine information.

The following audit classes could be established in **/etc/security/passwd**:

```
dave:
    auditclasses=general,config,system
mary:
    auditclasses=general,export
bill:
    auditclasses=general
```

Information Collection

AIX provides two different mechanisms by which the audit records may be collected and stored:

Bin Collection With the Bin collection interface, the audit records are written into two files that serve as temporary bins. After one bin is filled, records are written into the other bin while the first bin is being processed. After the processing is complete, the first bin is again available for use.

The **auditbin** program is a daemon which manages the files used for bin storage. As each bin is filled by the kernel, **auditbin** will run a configurable list of programs (contained in **/etc/security/audit/cmds**) to process the information in that bin. After these programs finish, **auditbin** notifies the kernel that the bin is available for further use.

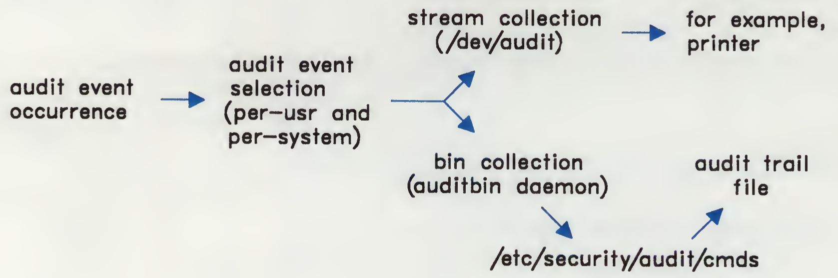
Stream Collection With the Stream collection interface, audit records are written into a circular buffer within the kernel itself. These records may be retrieved through a device interface (**/dev/audit**).

The system can be configured to log audit records using either interface or both interfaces simultaneously.

The Bin interface provides a more efficient interface for long term storage of a large audit trail. The programs which process bins supplied by the **auditbin** daemon usually append these bins to an audit trail file.

The Stream interface allows for highly interactive processing. Programs that process streams of audit records from **/dev/audit** usually display these records (for interactive viewing of auditable occurrences) or prints them (to provide a paper audit trail).

The following illustration shows the differences between Bin collection and Stream collection.



A5ACG039

Configuring the Bin Collection Interface

To configure the system to use the Bin collection interface, edit the **audit** stanza in the **/etc/security/config** attribute file. This stanza contains the following attributes:

```
binmode = {on|off|panic}
binsize = number
cmds = pathname
```

The **binmode** attribute controls how and whether the Bin collection interface is used. If the value of this attribute is **off** (the default), the Bin collection interface is not used. If the value is **on** or **panic**, the auditbin manager, **auditbin**, is started. When **binmode** is set to **panic**, then the kernel will panic if an error occurs when appending records into one of the bins. This mode should be specified if the system must not be used unless the auditing system is working properly.

The **cmds** attribute names a file that contains one or more shell command lines. The **auditbin** program will run each command on each bin that the kernel fills.

The **binsize** attribute specifies the size of the bin file.

Example Configuration of the Bin Collection Interface

Two configuration files, **/etc/security/config** and **etc/security/audit/cmds**, must be updated to provide for storage of a long-term audit trail. The following could be found in **/etc/security/config**:

```
audit:
        binmode=on
        binsize=20480
        cmds = /etc/security/audit/cmds
```

Where Bin collection, the size of the bins, and and the file containing the audit commands are specified.

The following could be found in **/etc/security/audit/cmds**:

```
/etc/auditapp -o /usr/spool/audit/trail $bin
```

Where **-o** precedes the path name of the trail to which records are to be appended.

Configuring the Stream Collection Interface

The Stream collection interface is enabled implicitly by running commands that open and read records from the audit device (**/dev/audit**). These commands should be placed in the **/etc/rc** commands file after the command that enables the auditing subsystem.

AIX provides the **auditstream** command, which will open the audit device and write audit records to its standard output. These records can be filtered with the command **auditselect**, displayed with the command **auditpr**, or simply redirected to a file. It is possible to have several programs reading independent streams of records simultaneously.

Example Configuration of the Stream Collection Interface

To maintain a hard-copy record of users of the system and their access rights, the following command can be added to **/etc/rc**:

```
auditstream | \
    auditselect "event == su || event == login || event == logout" |
    auditpr -v \
    > /dev/lp
```

Where **auditselect** specifies occurrances of **su**, **login**, and **logout** to be selected for the printed tail. See *AIX Operating System Commands Reference* for a discussion of these commands. See also "Audit Commands" on page 6-23 in this book.

Trail Analysis

An audit trail file can be analyzed using the **auditselect** command (an AIX audit trail reduction tool). This command reads the file and selects records that fit certain criteria. Using this tool, the administrator can select records based upon some combination of the event name, the user ID, the process ID, the time of the event, and other factors. The administrator is able to audit individual users, trace the actions of specific processes, or find all occurrences of a specific action in the audit trail.

For more information on the **auditselect** command, see *AIX Operating System Commands Reference.*

Enabling and Disabling the Auditing Subsystem

The auditing subsystem is enabled or disabled using the **audit** command. The **start** argument will read the audit configuration information, initialize the auditing subsystem, and turn auditing on. The **on** argument will simply turn auditing on without reconfiguring the subsystem. Finally, the **off** argument will turn auditing off.

The command:

```
audit start
```

should be inserted in the **/etc/rc** command file after you configure the auditing system so that auditing is enabled upon system initialization.

The audit system configuration can be changed or reset while the system is running, but you should not change the audit class definitions while users are logged in. Any running programs or processes affected by the change will produce unpredictable results.

You can reconfigure the auditing subsystem by

```
audit off
 . . . change the configuration information  . . .
audit start
```

Special Considerations: Extending the Auditing Subsystem

The AIX auditing subsystem is designed to be easily extended, so applications that do their own auditing can be added to the system. This is called trusted process auditing.

Programs should do their own auditing if they are privileged and use their privilege to change the security state of the system. Programs should also do their own auditing if they commit many auditable actions.

Trusted Process Auditing

To do its own auditing, a program must be privileged; that is, it must be installed as a setuid-root program. A program that does its own auditing should disable process auditing (using the **auditproc()** system call), so that the audit system will not record events detected in the kernel. Then the program should log its audit event with either the **auditwrite** command or the **auditlog()** system call. Both of these require the name of the event and the return status of the event. Event-specific information can be included in the record generated by trusted process auditing.

If the event being logged is already defined, then the optional information (sometimes referred to as the audit tail) must be in the format defined for that event in the

configuration file **/etc/security/audit/events**. No configuration changes are necessary. If it is a new event, however, the auditing subsystem must be configured to support it.

Adding Audit Events to the System

An audit event is defined symbolically as a 16 character string. This string must include the NULL terminating character. To add an audit event to the system, the administrator need only add that event to the file **/etc/security/audit/events**. This file contains definitions for all the audit events detected in the system. Each event is given as an attribute definition:

`eventname =` *formatting specification*

The formatting specification is used by the **auditpr** program to format the audit tail associated with a record.

Note: The primary purpose of adding the event name to this file is to communicate the formatting information to the program that prints audit records. This file also provides a comprehensive list of all the events detected by the system. This list should be kept complete and up to date so that when defining the audit classes, the person who administers the system can consult this list to be sure that all appropriate events are properly audited.

Other System Log Files

Some system commands also store records of their actions in other files that are not part of the auditing system.

/etc/.ilog

Invalid login attempts caused by incorrect login names or passwords are recorded in the **/etc/.ilog** file. The contents of this file can be examined only by a privileged user or a member of the system group.

Each time you log in as **root** or execute **su** and there is an entry in **/etc/.ilog**, the system displays a message advising you to check the contents of **/etc/.ilog**. It is important for security reasons to keep track of invalid login attempts since an invalid login attempt may indicate an attempted security breach. Use the **who** command to read the contents of this file:

```
who /etc/.ilog
```

Successful Uses of su

Successful invocation of the **su** program causes a record to be appended to the **/usr/adm/sulog** file, if it exists. This is an ASCII file which may be displayed: for example, by the **cat** program.

Part 2. Managing AIX Data Communications

Chapter 7. Managing the Electronic Mail System

CONTENTS

About This Chapter

Note: Before reading this chapter, read and understand the information about using the mail system in *IBM RT Using the AIX Operating System.* This information includes descriptions of how the mail system handles messages and of mail addressing formats, and provides an essential background for understanding the information in this chapter.

This chapter describes the general structure of the electronic mail system and how to set it up to route mail in local and networking configurations. To use the mail system to route mail in a network, you must have already installed the network and have it working for other network functions. Types of networks over which you can route mail include:

- **TCP/IP** - Refer to *Interface Program for use with TCP/IP* for information to configure and use this network.
- **uucp** - Refer to Chapter 9, "Managing the Basic Networking Utilities" on page 9-1 for information to configure this network. Refer to *IBM RT Using the AIX Operating System* for information to use this program on an asynchronous network.

This chapter also includes information to help you manage the mail system including information about:

- Managing the mail queue
- Keeping a record of mail system activities
- Changing the configuration file to meet unusual circumstances.

Refer to *AIX Operating System Commands Reference* for reference information about all options available for the commands discussed in this chapter. Refer to *AIX Operating System Technical Reference* for additional information about the configuration file **sendmail.cf**.

Understanding the Electronic Mail System

The mail system is a general purpose, inter-network mail routing facility. It is not tied to any one transport protocol. It relays messages from one user to another across system and network domain boundaries. While processing the messages, the mail system can do a limited amount of message header editing to put the message into a format that is appropriate for the receiving domain.

The mail system consists of the following main components as shown in Figure 7-1.

- A user interface program for handling messages
- The **sendmail** program for routing messages
- One or more mailer programs to transmit messages to their destinations.

The following paragraphs outline the functions of these components.

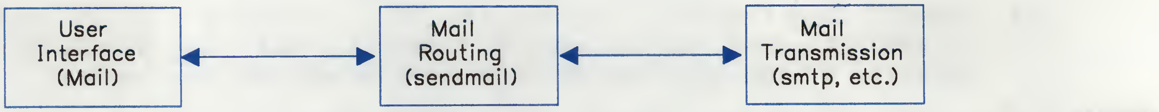


Figure 7-1. Mail System Overview

The User Interface

The user interface to the mail system is the **mail** program. This program allows the user to:

- Compose messages to send to other users
- Receive and read messages sent by other users
- Manage a set of mail box and folder files to keep track of messages sent and received.

Refer to *IBM RT Using the AIX Operating System* for information about using **mail**. This program is provided as the standard interface to the mail system. Another program, the message handler (MH), is also available as an installed option to replace this user interface program (MH can also be used together with **mail**). Refer to Chapter 10, "Overview of the Message Handling Package" on page 10-1 and *AIX Operating System Commands Reference* for information about this program.

Mail Routing

The **sendmail** program provides mail routing functions for the mail system. This program interfaces with several user application programs to receive and deliver messages to users. In addition, you can configure it to route mail across network connections to users at remote systems. The **sendmail** program can route mail to users in the following network situations:

- Users on the local system
- Users connected to the local system with a TCP/IP protocol
- Users connected to the local system with a **uucp** protocol.

Refer to *IBM RT Using the AIX Operating System* for a description of the addressing format used to deliver mail in any of these situations.

Mailer Programs

Mailer programs are those programs that either deliver the message to local users or transmit the message to a remote system. The programs that perform these functions include:

bellmail This program delivers mail to local users.

sendmail This program uses SMTP to transmit mail across a TCP/IP network.

uucp This program transmits mail across an asynchronous connection.

Mail Transmission

In AIX the Simple Mail Transfer protocol (SMTP) supports sending and receiving mail (messages) between users on different hosts. The **sendmail** command implements this protocol and supports the following SMTP commands:

DATA
EXPN
HELO
MAIL
NOOP
QUIT
RCPT
RSET
VRFY

Refer to **sendmail** in *AIX Operating System Commands Reference* for more information on the **sendmail** server command, to *IBM RT Using the AIX Operating System* for more information on mail, and to RFC 821 for more information on SMTP.

Understanding Mail System Files

When you install **sendmail** and set up a mail system for your computer, you must install, create or change some files in the system. Figure 7-2 on page 7-8 shows the files associated with the mail system. The following paragraphs summarize the purpose of the files that are affected by setting up a mail system.

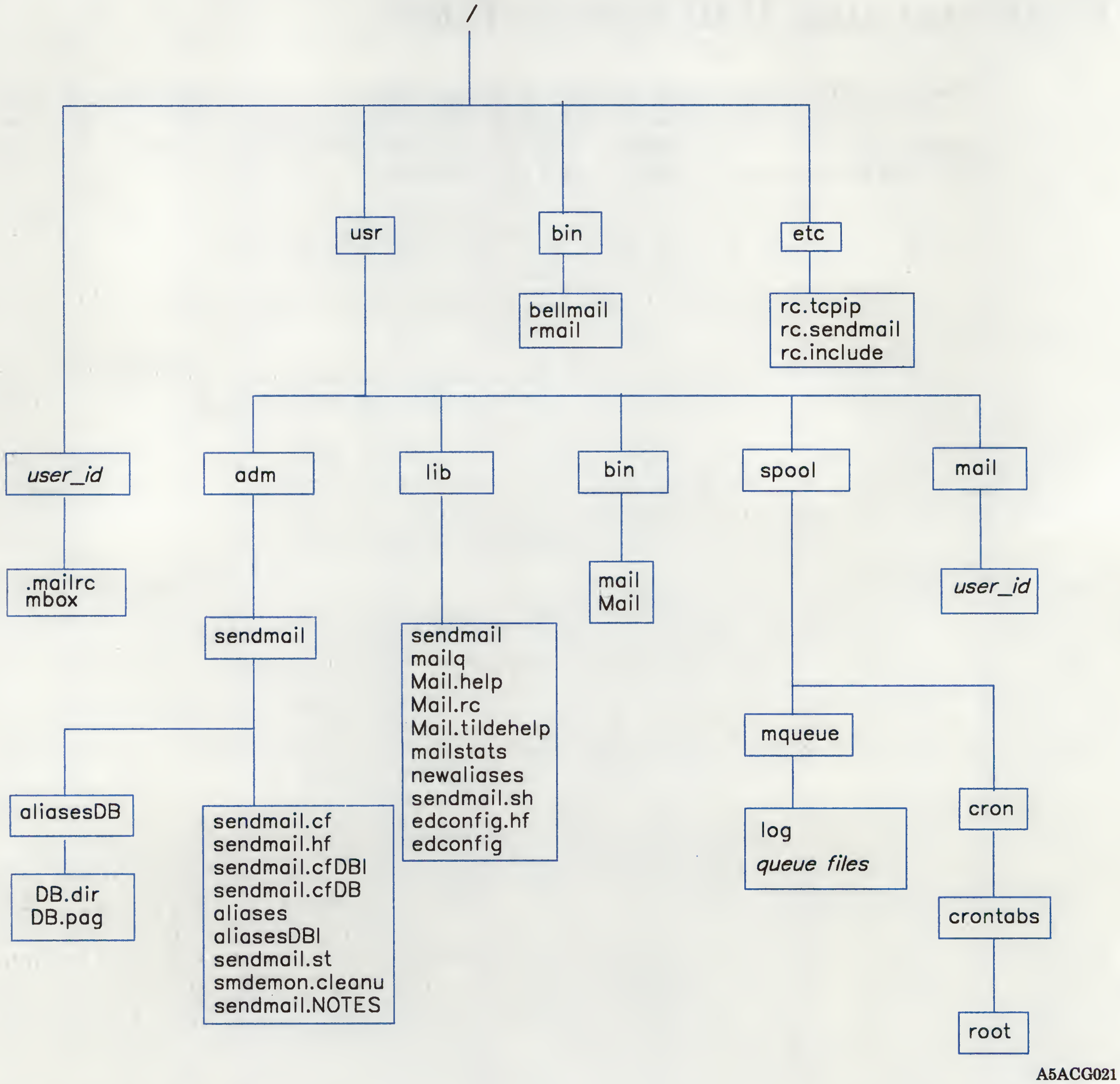


Figure 7-2. Mail System Files

Understanding Files for the Mail Command

The **mail** command is the user interface to the mail system. Refer to *IBM RT Using the AIX Operating System* for a description of how to use this program. The following files are associated with the **mail** program:

Directory or File	Use
/usr/lib:	
Mail.help	This file is a text file containing help information for using the mailbox handling function of the **mail** program.
Mail.tildehelp	This file is a text file containing help information for using the **mail** editor to create messages.
Mail.rc	This file is a text file that you can modify to set the default characteristics of the **mail** program (see "Defining the Characteristics of the Mail Program" on page 7-22).
/u/*user_id***:**	
.mailrc	This file is a text file that each user creates in the **$HOME** directory. It allows the user to change the system defaults for the **mail** program when that user runs the program. See *IBM RT Using the AIX Operating System* for information about this file.
mbox	This file is a text file that stores processed mail for the individual user. See *IBM RT Using the AIX Operating System* for more information about this file.
/usr/bin:	
Mail or **mail**	These two names are linked to the same program. The **mail** program is the user interface to the mail system.
/usr/mail	

Figure 7-3 (Part 1 of 2). Mail Files

Directory or File	Use
user_id	This directory stores all incoming mail for users. It contains a file for each user that has incoming mail using the user ID for the name of the file.
/bin:	
bellmail	This file contains the original mail program that comes with the operating system. This program performs local mail delivery.
rmail	This program is the interface to **sendmail** that **uucp** uses to deliver mail. It is not the same **rmail** program that existed in earlier versions of the operating system.

Figure 7-3 (Part 2 of 2). Mail Files

Understanding Files for the sendmail Program

Although the **sendmail** program can be used directly to send mail to other users, it is normally used as a background process to control mail routing. The **sendmail** program uses the following files. You can use different file names for many of these files by specifying a flag or configuration option to **sendmail**. See *AIX Operating System Commands Reference* for complete information about **sendmail** command flags.

Directory or File	Use
/usr/lib:	
sendmail	This file is the **sendmail** program.
mailq	This file is a link to **/usr/lib/sendmail** that executes **/usr/lib/sendmail -bp** to print a listing of the mail queue.
newaliases	This file is a link to **/usr/lib/sendmail** that executes **/usr/lib/sendmail -bi** to build an alias data base from the text file **/usr/adm/sendmail/aliases**.
mailstats	This command formats and prints the **sendmail** statistics as found in **/usr/adm/sendmail/sendmail.st** if that file exists (see "Logging Mailer Statistics" on page 7-27).
sendmail.sh	This file is a shell program that acts as the **sendmail** program when the **sendmail** program is not installed. It acts as an interface to the **bellmail** program for local mail delivery. It is installed as **/usr/lib/sendmail** with the base operating system. When **sendmail** is installed, this program is moved to **/usr/lib/sendmail.sh** and **sendmail** is loaded to **/usr/lib/sendmail**.
edconfig	This program provides a menu interface for editing many parameters in the **sendmail** configuration file.
edconfig.hf	This file contains the help text that the **edconfig** program displays.
/usr/adm/sendmail:	
aliases	This file is a text version of the aliases file for **sendmail**. You can edit this file to create new aliases for your system.
aliasesDB	This directory contains the aliases database files, **DB.dir** and **DB.pag** that are created from **/usr/adm/sendmail/aliases** when you run **sendmail -bi**.

Figure 7-4 (Part 1 of 3). Sendmail Files

Directory or File	Use
aliasesDBl	This is a lock file for the **aliasesDB** data base.
sendmail.hf	This file contains help information used by the **smtp HELP** command.
sendmail.cf	This file contains the **sendmail** configuration information in text form. You can edit this file to change this information (see "Changing the Sendmail Configuration File" on page 7-35).
sendmail.cfDB	This file contains the processed version of the **sendmail.cf** file. This file is created from **sendmail.cf** when you run **/usr/lib/sendmail -bz**.
sendmail.cfDBl	This is a lock file for the **sendmail.cfDB** data base.
smdemon.cleanu	This file is a shell program file that runs the mail queue and maintains the **sendmail** log files in the directory **/usr/spool/mqueue**. See "Managing the Log and Mail Queue" on page 7-25 for more information about this program.
sendmail.NOTES	This is an informational file. It contains information regarding the queue directory cleaning facility, and sendmail queue cleaning facility.
sendmail.st	This file collects statistics about mail traffic. The file does not grow. Use the **/usr/lib/mailstats** command to display the contents of the file. You do not need to have this file if you do not want to collect this information. See "Logging Mailer Statistics" on page 7-27 for more information.
/usr/spool:	
mqueue	This directory contains the **log** file and temporary files associated with each message in the queue. Refer to "Managing the Mail Queue" on page 7-29 for information about the temporary queue files. Refer to "Logging Mail System Activities" on page 7-23 for information about the mail log file.

Figure 7-4 (Part 2 of 3). Sendmail Files

Directory or File	Use
cron/crontabs	This directory contains files that **cron** reads to determine jobs that it is to start. The **root** file contains a line to start **smdemon.cleanu** (see "Managing the Log and Mail Queue" on page 7-25).
/etc:	
rc.include	This file contains invoication for **rc.tcpip** which is added when **TCP/IP** is installed. This file should be uncommented.
rc.tcpip	This file contains shell scripts to start up many functions associated with TCP/IP. If you install TCP/IP, you must uncomment lines in this file to start **sendmail**. See "Starting sendmail with TCP/IP Installed" on page 7-20 for information about changing this file.
rc.sendmail	This file contains a shell script to start the **sendmail** daemon.

Figure 7-4 (Part 3 of 3). Sendmail Files

Setting up Mail Delivery

You can set-up the mail system to deliver mail to users on a local computer, to users connected by a local area network, to users connected by a **uucp link**, or a combination of these connections. Setting up the mail system for any of these environments requires similar procedures for each type of configuration. Only the information provided to **sendmail**, and the type of addressing required for each network are different.

Setting up Mail Delivery

Note: You must be a member of the system group to configure the mail system.

1. Install the **sendmail** program from the Extended Services Program (see *Installing and Customizing the AIX Operating System*).
2. Ensure that the configuration information in **/usr/adm/sendmail/sendmail.cf** accurately reflects your system's needs. See "Defining Mail Requirements" on page 7-15 for information about this file. Then, build a data base version of the configuration file:

   ```
   /usr/lib/sendmail -bz
   ```
3. Define the addressing and routing information required for your mail delivery system. See "Defining Addressing and Routing Information" on page 7-16 for information about addressing requirements.
4. Ensure that the alias information in **/usr/adm/sendmail/aliases** accurately reflects your system's needs. See "Defining Aliases" on page 7-17 for information about this file. Then, create data base files from this file:

   ```
   /usr/lib/sendmail -bi
   ```
5. Change the proper system start-up file to include code that starts the **sendmail** daemon (see "Starting the sendmail Daemon" on page 7-19) so that the daemon will be started automatically each time the system is started.
6. Start the **sendmail** daemon:

   ```
   sh /etc/rc.sendmail
   ```

Defining Mail Requirements

The **sendmail** program reads the file **/usr/adm/sendmail/sendmail.cf** for information about the network addressing, the type of mailer programs being used and information about how it should format messages. This file is supplied with the **sendmail** programs and contains a set of information that allows **sendmail** to operate in the following environments with little or no changes required to the file:

- Local mail delivery
- Local area network delivery using TCP/IP addressing formats:
  - *user_id*
  - *user_id@host*
  - *user_id@host.domain*
- Remote delivery using **uucp**.

If your environment includes one or all of these types of mail delivery, you may be able to use the supplied configuration file with only a few changes:

- Host name macro (see "Changing the Host Name Macro" on page 7-37)
- Host name class (see "Changing the Host Name Class" on page 7-38)
- Domain name macro (see "Changing the Domain Name Macro" on page 7-39)
- Domain name classes (see "Changing the Domain Name Part Macros" on page 7-39).

Only unique situations should require any other changes to this file. For information about changing this file, refer to "Changing the Sendmail Configuration File" on page 7-35.

Building the Configuration File

The first time you install **sendmail** and each time you change the configuration file, you must build the data base version of the configuration file:

```
/usr/lib/sendmail -bz
```

This operation creates the file **/usr/adm/sendmail/sendmail.cfDB** which contains the data base version of the configuration information. If the **sendmail** daemon is running, you must kill that process and start the daemon again (see "Starting the sendmail Daemon" on page 7-19) for the new configuration file to take effect.

Defining Addressing and Routing Information

The type of routing and addressing information needed depends upon how complex your mail delivery system is. Mail can be delivered:

- Locally
- To users on a local area network
- To users on a **uucp** network.

To deliver mail in any of these modes, you must define the addressing and routing information needed for the selected mode. To deliver mail in more than one mode, you must define the addressing and routing information for each mode that you want to use.

Defining Local Information

To deliver mail in a local environment (where all senders and receivers are users on the same system), no additional addressing and routing information is required. The **sendmail** program uses the information already available in the local system files to determine the correct routing for a local message.

Defining Local Area Network Information

To deliver mail in a local area network (where senders and receivers may be on different systems connected by local area network links), the network must be using TCP/IP as one available network protocol. For networks with many systems, you may want to set-up one or more systems on the network as the name server for the network. The name server controls the addressing information for the network and for connections to systems beyond the immediate network. Using a name server simplifies the task of keeping the address information up-to-date. See *Interface Program for use with TCP/IP* for information about using this program and for instructions to set up a system as a name server on a local area network.

Defining uucp Information

To deliver mail to another system connected to the local system with a **uucp** link, you must define the addressing and routing information required for that link (see Chapter 9, "Managing the Basic Networking Utilities" on page 9-1). The **sendmail** program transfers mail to the **uucp** daemons (through the **uux** command) for delivery across a **uucp** link. The **uusched** program handles the queued routing of mail for **uucp** links.

Defining Aliases

A mail system ***alias*** is a name that represents an address or set of addresses. Purposes of defining aliases include:

- **Convenience**: Use a short name, such as `mark` to represent a long address such as `mark@zeus.aus.IBM.COM`.
- **Accuracy**: Using a short name for a complex address helps reduce the chance of a keying error in the address.
- **Distribution Lists**: Using a short name for a list of addresses allows you to send the same message to many people using only one address when you compose the message.
- **Rerouting Mail**: You can cause mail for one user ID to be delivered to another user ID with an alias.
- **Sending Mail to a File**: You can cause mail for a user ID to be delivered to a file instead of the system mailbox by using an alias.

You can define two types of aliases: personal aliases and local-system aliases. Personal aliases are aliases that operate on mail produced by the individual user that defined the alias. These aliases are defined in the user's **.mailrc** file as described in *Using the AIX Operating System.* Local-system aliases are aliases that apply to all mail handled by **sendmail** on the local system. Define these aliases in **/usr/adm/sendmail/aliases**, which is an editable text file that you can change if you are a member of system group.

The **aliases** file is included with the files that you install with the **sendmail** program. This file has all required aliases defined so that **sendmail** will work without changing this file in many cases. You may have to change this file if the predefined aliases do not match your system configuration, or to add new local-system aliases. After you change the **aliases** file, you must rebuild the alias data base that **sendmail** actually uses. If you do not rebuild the data base, **sendmail** does not recognize your changes.

Building the Alias Data Base

The **sendmail** program does not use the alias definitions in the **aliases** file directly. Instead, you must process this file to produce data base files that allow **sendmail** to process them more quickly. When you are satisfied that the **aliases** file contains all the information that you need for your system, use the following command to create the alias data base:

```
/usr/lib/sendmail -bi
```

This command causes **sendmail** to read the **aliases** file and create two additional files containing the alias data base information. These new alias files are:

- **/usr/adm/sendmail/aliasesDB/DB.dir**
- **/usr/adm/sendmail/aliasesDB/DB.pag**

If these files do not exist, **sendmail** cannot start and **sendmail** generates an error message when it tries to start.

Creating a New Alias

To create a new local-system alias, edit the file **/usr/adm/sendmail/aliases**. You must be a member of system group to edit this file. Aliases in this file are defined with an entry in the following format:

a_name: *name1*, *name2*, ...*namex*

In this format *a_name* can be any alphanumeric string that you choose (not including any special characters, such as @ or !), and *name1* through *namex* is a series of one or more addresses. To continue the list of names on one or more lines, begin each continued line with a space or a tab. Blank lines and lines beginning with a # (number sign) are comment lines that are not included in the alias data base.

For example, the following entry defines an alias `writers` to be a set of addresses of people in that group:

```
writers: geo, mark@zeus, ctw@athena, brian
```

This definition could be contained on several lines also as long as each added line begins with a space or a tab:

```
writers: geo,
       mark@zeus,
       ctw@athena,
       brian
```

Required Aliases

The **aliases** file must contain alias definitions for the following names:

MAILER-DAEMON

Assign this name to the ID of the user that is to receive mailer daemon messages. This name is initially assigned to `root`:

```
MAILER-DAEMON: root
```

postmaster

Assign this name to the ID of the user responsible for the operation of the local mail system. The alias `Postmaster` (in any combination of upper- or lower-case letters) defines a single mailbox address that is valid at each system in a network. This address allows users to send inquiries to the Postmaster at any system without knowing the correct address of any user at that system. This name is initially assigned to `root`:

```
postmaster: root
```

nobody Assign this name to the ID that is to receive messages directed to programs such as **news** and **msgs**. This name is initially assigned to **/dev/null**:

```
nobody: /dev/null
```

Special Alias for List Owners

When creating a distribution list alias, you should also assign one ID as the *owner* of that list. If **sendmail** has a problem sending mail using a certain list, it sends an error message to the owner of that list. The format for assigning an owner to a list is:

`owner-`*listname*: *owner_addr*

In this format, *listname* is the alias name of the distribution list and *owner_addr* is the address to which to send error messages for that distribution list. For example, the following set of entries in **aliases** defines a distribution list, `editors`, and its owner `glenda@hera`:

```
editors: glenda@hera, davidm@kronos, perryw@athena
owner-editors: glenda@hera
```

If there is no ID for `perryw` on system `athena`, then **sendmail** sends an error message to the mailbox for `glenda@hera` when someone sends mail to the `editors` distribution list.

Starting the sendmail Daemon

The **sendmail** daemon listens on the **smtp** socket for connections (to receive mail from a remote system) and processes the queue periodically to ensure that mail gets delivered when remote systems become active. Using flags to the **sendmail** command, you can activate either or both of these functions when you start the daemon.

The file **/etc/rc.sendmail** contains the program statements that start the **sendmail** daemon. Depending upon your configuration, you can run this file in one of the following ways:

- Manually from the command line, using the **sh** command:

  ```
  sh /etc/rc.sendmail
  ```

- Automatically from the file **/etc/rc.tcpip** (if TCP/IP is installed and running)

Use the manual method only as part of the first installation procedure. Afterwards, start **sendmail** automatically.

Starting sendmail with TCP/IP Installed

If TCP/IP is installed, the system runs the file **/etc/rc.tcpip** when it starts. This file contains a section of shell commands that runs the **sendmail** start-up file (**/etc/rc.sendmail**). Initially, this section is prevented from running because it has # (comment characters) at the beginning of the lines. The section looks like the following:

```
# Start up Sendmail Daemon
# if [ -f /usr/lib/sendmail ]; then
#    sh /etc/rc.sendmail 1>>/dev/null 2>&1 &
#    echo " sendmail\c" >/dev/console
# fi
```

To start the daemon when the system starts, edit the file and remove the last four comment characters from this section:

```
# Start up Sendmail Daemon
  if [ -f /usr/lib/sendmail ]; then
     sh /etc/rc.sendmail 1>>/dev/null 2>&1 &
     echo " sendmail\c" >/dev/console
  fi
```

Save the file. The next time the system starts, **sendmail** will start as a background process.

Mail Delivery without TCP/IP Installed

Mail is delivered automatically to users on the local system without running the **sendmail** daemon. However, since mail can still enter the **sendmail** queue, you should process the queue periodically to ensure that that mail is delivered. See "Managing the Log and Mail Queue" on page 7-25 for information about using **cron** to force delivery of all mail in the queue. You can also process the mail queue by entering the following command on the command line:

```
# /usr/lib/sendmail -q
```

If your system is connected to others over **uucp** links (see Chapter 9, "Managing the Basic Networking Utilities" on page 9-1), the mail system uses **uux -r** to place the mail for those systems in the **uucp** spooling directory. After that, **uucp** controls delivery of the mail to the remote systems.

Starting sendmail for Debugging Purposes

For special cases, such as testing and debugging a new installation, you may want to start **sendmail** directly from the command line using the command line options described in *AIX Operating System Commands Reference.*

The **sendmail** program includes debug flags. Each flag has a number and a level. Higher levels print out more information. The only debug flag that is useful without **sendmail** source code is flag 21 (for debugging rewriting rules). You must be a member of system group or know the **sendmail** debug password (set with the **-bw** flag to **sendmail** - see *AIX Operating System Commands Reference*). Set debug flags with the **-d** flag to **sendmail**. The syntax is:

`/usr/lib/sendmail -d`*debug-flag*`[.`*level*`]`

-or-

`/usr/lib/sendmail -d`*passwd* `-d`*debug-flag*`[.`*level*`]`

In this format *debug-flag* is an integer that specifies the built-in debug flag to be set, and *level* is an optional integer that specifies the debug level defined for that flag. Imbedded blanks are not allowed. If you do not specify *level*, **sendmail** uses level 1. You can specify more than one **-d** flag. If the debug password is required, provide it as the argument to the first **-d** flag. You can specify more than one *debug-flag.level* pair by separating them with a comma:

debug-flag.level,*debug-flag.level* ...

where spaces are for reading ease only. You can also specify a range of flags to set using a dash (-) to separate the beginning and the ending flag numbers for the range. For example:

```
-d12                Set flag 12 to level 1
-d12.3              Set flag 12 to level 3
-d3-17              Set flags 3 through 17 to level 1
-d3-17.4            Set flags 3 through 17 to level 4
```

Defining the Characteristics of the Mail Program

You can define the characteristics of the **mail** program to provide a basic behavior of the program from which the rest of the system users can build. The **mail** program reads the file, **/usr/lib/Mail.rc** each time someone uses the program. This file can contain any of the mail commands for defining:

- Aliases
- The prompts that the program provides
- Normal disposition of mail files
- Other operational characteristics of the **mail** program.

These commands are the same as the commands that each user can put in **.mailrc** as described in the chapter of *IBM RT Using the AIX Operating System* that discusses the mail system. Refer also to *AIX Operating System Commands Reference* for other commands that can be used in this file.

Remember that commands in **Mail.rc** override the default characteristics of the **mail** program for all users on the system. Users can create a **.mailrc** file in their home directories to contain commands to customize the **mail** program for their own use. These commands override similar commands in **Mail.rc**. In addition, users can enter the commands while using the **mail** program to override similar commands in both **.mailrc** and **Mail.rc**.

Logging Mail System Activities

The **sendmail** program can automatically write messages into a log file when activities occur that affect the mail system. The log file is named **log** in the mail queue directory (**/usr/spool/mqueue** unless you change it - see "Setting Configuration Options" on page 7-40). You must be a member of system group to access this file.

Understanding the Log Format

Messages in the log file appear in the following format:

<*n*> *Mmm dd hh*:*mm*:*ss* [*pid*] (*uname*) *msg_text*

The symbols in this format have the following meanings:

Symbol	Meaning
n	This field contains an integer between 0 and 7 that specifies the severity or importance of the message. The most important is 0; the least important is 7. Valid numbers are: **0** The system cannot be used. **1** Take action quickly to correct the problem. **2** The message represents a critical condition. **3** The message represents an error condition in the mail system. **4** The message represents a warning of a condition that could cause a problem. **5** The message informs you of a normal, but significant, condition. **6** The message represents a normal activity. **7** The message contains information for debugging the **sendmail** program.

Figure 7-5 (Part 1 of 2). Mail Log Fields

Symbol	Meaning
Mmm dd hh:mm:ss	These fields specify the time that the message was entered in the log file: *Mmm* A 3-letter abbreviation of the month *dd* The date of the month (01-31) *hh* The hour of the day (00-23) *mm* The minute of the hour (00-59) *ss* The second of the minute (00-59)
pid	This field contains the process ID of the process that generated the message.
uname	This field contains the user name associated with the message.
msg_text	This field contains a brief explanation of the condition that caused the message to be written to the log file.

Figure 7-5 (Part 2 of 2). Mail Log Fields

For example, the following entry occurred in a log file when the system was started up, indicating that the **sendmail** program was started:

```
<6> Jan  7 07:43:49 [116] (root) Daemon/queue proc started, pid 116
```

Choosing a Log Level

You can choose the types of activities that **sendmail** puts into the log file by changing the **L** option line in **sendmail.cf** (see "Changing the Operational Logging Level" on page 7-41 for information about changing this option). If you change the configuration file, you must process it with the **sendmail -bz** command before **sendmail** recognizes the change. In the standard configuration file the level is set with the following statement:

```
OL9
```

This statement turns on the logging function to level 9. Valid levels and the activities that they represent are (each number includes the activities of all numbers of lesser value and adds the activity that it represents):

Level	Activity Logged
0	Major activities only (building a configuration file or creating an alias database)
1	Major problems only

2	Message collections and failed deliveries
3	Successful deliveries
4	Messages being deferred (due to a host being down, etc.)
5	Placing messages in the queue (normal event)
6	Unusual but benign incidents (trying to process a locked file, etc.)
9	Log internal queue id to external message id mappings. This can be useful for tracing a message as it travels between several hosts.
12	Several messages that are of interest when debugging.
16	Verbose information regarding the queue.

If you change this file, you must process it with the **sendmail -bz** command before **sendmail** recognizes the change.

Managing the Log and Mail Queue

Because information is continually appended to the end of the log file, it can grow to become very large. Also, error conditions can cause unexpected entries to the mail queue. To help prevent problems associated with the mail queue and log, use the program **/usr/adm/sendmail/smdemon.cleanu**. This program forces **sendmail** to process the queue, and also maintains four progressively older copies of log files, named **log.0**, **log.1**, **log.2**, and **log.3**. Each time the program runs it moves:

1. **log.2** to **log.3**
2. **log.1** to **log.2**
3. **log.0** to **log.1**
4. **log** to **log.0**

This action removes the **log** file, allowing logging to start over with a new file. You can change this program to create more or fewer aging copies of the log file. Use the following procedure to have **cron** start this program at a specified time:

1. Log in as root, and change to the **crontabs** directory:

   ```
   cd /usr/spool/cron/crontabs
   ```

2. Edit the file **root** with your favorite editor.
3. The file contains a line similar to the following line:

   ```
   #45 23 * * * ulimit 5000; /usr/adm/sendmail/smdemon.cleanu > /dev/null
   ```

Remove the # at the beginning of that line. This entry runs the program every day at 11:45 PM. If your system is not operating at that time, change the second number (23 representing the 23rd hour, or 11:00 PM) to a time when your system is on. For example, to run the cleanup program at 4:45 PM:

```
45 16 * * * ulimit 5000; /usr/adm/sendmail/smdemon.cleanu > /dev/null
```

4. Save the file and exit from your editor.
5. Notify **cron** of the change to the file by reloading the file while still in the **crontabs** directory:

   ```
   crontab root
   ```

Logging Mailer Statistics

The **sendmail** program can keep track of the volume of mail being handled by each of the mailer programs that interface with it (those mailers defined in **sendmail.cf** - see "Defining a Mailer" on page 7-52). To start the accumulation of mailer statistics, create the file **/usr/adm/sendmail/sendmail.st** (you must be a member of system group to create this file):

```
touch /usr/adm/sendmail/sendmail.st
chmod 660 /usr/adm/sendmail/sendmail.st
```

When this file exists and is either empty or contains a correctly formatted data structure, **sendmail** keeps track of mail volume handled by each mailer in a data base in that file. If the file is not in the correct format, **sendmail** does not change the file and does not log mailer statistics. This may happen if a statistics file created by an older version of **sendmail** exists on the system. In that case, save the older file for processing by the appropriate older version of **mailstats**. Then you can either remove the present file and create a new statistics file as shown above, or clear the statistics file using **mailstats -z** to begin recording statistics.

If **sendmail** encounters errors when it tries to record statistics information, it writes a message on the system console and in the **sendmail** log file. These errors do not affect other operations of **sendmail**.

The **sendmail** program updates the information in the file each time that it processes mail. The size of the file does not grow, but the numbers in the data base do. They represent the mail volume since the time you created or reset **sendmail.st**. To reset the numbers and start counting over again, use the **-z** flag to the **mailstats** command:

```
/usr/lib/mailstats -z
```

Displaying the Mailer Information

The information kept in **sendmail.st** is in a data base format that cannot be read as a simple text file. To display the mailer statistics, use the **mailstats** command:

```
/usr/lib/mailstats
```

This command reads the information in **sendmail.st**, formats it, and writes it to standard output. The format of the information is shown in the following example:

```
Sendmail statistics from file "/usr/adm/sendmail/sendmail.st"
           Collection started at Thu Feb 18 17:40:41 1988

Mailer           msgs_from     bytes_from     msgs_to      bytes_to
------------     ---------     ----------     ---------    ----------
local                    1              2             1           201
prog                     0              0             0             0
uucp                     0              0             0             0
tcp                      0              0             0             0
```

The fields in the report have the following meanings:

Mailer This field contains the name of the mailer program that handled the mail.

msgs_from
This field (*messages from*) contains the number of messages that originated with the indicated mailer.

bytes_from
This field contains the number of bytes of information in the messages received from the indicated mailer.

msgs_to This field (*messages to*) contains the number of messages that were sent out from **sendmail** using the indicated mailer.

bytes_to This field contains the number of bytes of information in the messages sent to the indicated mailer.

The collection start time indicated on the second line of the report is the time at which the first update to the empty file was performed.

If **sendmail** transmits mail directly to a file, such as **dead.letter** or an alias target, the message and byte counts are credited to the `prog` mailer in addition to the normal statistics for use of the `prog` mailer.

Statistics Messages

When **mailstats** is called with no program flags, it can generate the following messages:

`No statistics data in file "/usr/adm/sendmail/sendmail.st"`
The **sendmail** program has not written any data into the statistics file.

`mailstats: file size change; use previous mailstats version`
The statistics file format is not the format expected by **mailstats**. Try using a previous version of **mailstats** to read it.

Managing the Mail Queue

The **sendmail** program processes the mail queue automatically at intervals determined by a value that you provide to the **sendmail** daemon when it starts (see "Determining the Queue Processing Interval"). You can also process the mail queue by starting **sendmail** directly (or from **cron**) with the **-q** flag. When **sendmail** processes the queue, it:

1. Reads and sorts the queue
2. Processes the jobs in order on the queue:
    a. Checks to see if the job is locked.
    b. If it is not locked, **sendmail** processes the job.

Locking of the jobs allows more than one queue processor to be active at the same time. You do not need to wait for one queue processor to finish before starting another.

In most cases, **sendmail** manages the mail queue without need for you to get involved. However, for some unusual circumstances you may need to process the mail queue manually. For example, if a major system on your network is not operating for a long time, the queue at your system may become loaded with messages routed through that system. With a huge backlog of messages to deliver, performance on your system may go down. When the remote system recovers, **sendmail** delivers all the messages to that system. However, you may want to intervene to improve system performance until the remote system recovers (see "Moving the Queue" on page 7-34).

Determining the Queue Processing Interval

The interval at which the **sendmail** daemon processes the mail queue is determined by the value of the **-q** flag when the program starts. You start the daemon by running the shell program **/etc/rc.sendmail** which contains a default value for the queue processing interval (typically about 30 minutes). If this value does not suit your system, you can change it (if you are a member of system group):

1. Edit **/etc/rc.sendmail**.
2. Near the beginning of the file, find a line that assigns a value to the variable `qpi`, such as:

    ```
    qpi=30m
    ```

    This value indicates a queue processing interval of 30 minutes.

3. Change the value assigned to `qpi` to a number of minutes to suit your needs. You can also change the letter `m` to change the unit of time (minutes in this case) to hours or another unit. See "Specifying Time Values to Sendmail" for the format of values for this and other time parameters.

4. Save the file and exit the editor.

For the changes to have an effect, you must kill the current **sendmail** daemon and start a new one:

1. Use the **ps** command to determine the process ID of the current **sendmail** daemon.

2. Use the **kill** command to kill the current **sendmail** daemon.

3. Start a new **sendmail** daemon:

   ```
   sh /etc/rc.sendmail
   ```

Specifying Time Values to Sendmail

All time intervals are set using a number followed by a letter to designate the time unit being used. For example, **10m** represents ten minutes, where **2h30m** represents two hours and 30 minutes. If you provide a number without a letter to determine the units, **sendmail** uses *days*. For example, **10** represents ten days. The full set of scales is:

s	seconds
m	minutes
h	hours
d	days
w	weeks

Displaying the Mail Queue

Use the **mailq** command to print a listing of the current contents of the mail queue. The format of this command is:

```
/usr/lib/mailq
```

This command produces a listing of the content of the mail queue as shown in the following example listing:

```
                Mail Queue (1 request)
--QID-- -Size- -----Q-Time----- ------------Sender/Recipient-----------
AA00269      6 Thu Dec 17 10:01 geo
               (Deferred: Connection timed out during user open with zeus)
                                 amy@zeus
```

Figure 7-6 shows the meanings of the fields in this listing.

Field	Meaning
QID	This column contains the message queue ID of the message.
Size	This column contains the number of bytes in the text portion of the message (heading information not included).
Q-Time	This column contains the date that the message entered the queue.
Sender/Recipient	This column contains the user ID of the sender (first) and then the user ID of the receiver of the message. In this example, a message also appears to indicate the status of the message.

Figure 7-6. Mail Queue Fields

Examining the Message Queue Files

The **sendmail** program keeps mail queue files in the directory **/usr/spool/mqueue**. You must be a member of system group to work with this directory. Each message in the message queue has a number of files in this directory. The files are named with a prefix letter, the letter f and the message queue ID of the message in the following format:

typefID

In this format *ID* is the message queue ID, f is a fixed letter, and *type* can be one of the following letters that indicate the type of file it is:

d This is the data file containing the message body without the heading information.

l This is a lock file. This file may not be present. If it is present, the job is currently being processed and a queue run will not process the file. For that reason, an extraneous l file can cause a job to apparently disappear without timing out.

n This file is created when an ID is being created. Because it is present for only a short period of time, you may not see an **n** file unless an error occurred. It is a separate file to ensure that no mail can ever be destroyed due to a race condition.

q This is the queue control file. This file contains the information necessary to process the job (see "Examining the q File").

t This is a temporary file. This file is an image of the **q** file when it is being rebuilt. It is renamed to a **q** file very quickly.

x This is a transcript file that exists during the life of a session showing everything that happens during that session.

As an example, if a message has a queue ID of `AA00269`, then the following files are created and deleted in the mail queue directory while **sendmail** tries to deliver the message:

File	Function
`dfAA00269`	Data file
`lfAA00269`	Lock file
`nfAA00269`	Back up file
`qfAA00269`	Control file
`tfAA00269`	Temporary file
`xfAA00269`	Transcript file

Examining the q File

The **q** file contains a series of lines each beginning with a code letter. The code letters are as follows:

D This line contains the name of the data file. There can be only one of these lines.

H This line contains a heading definition. There may be any number of these lines. The order in which the **H** lines appear determines their order in the final message. These lines use the same syntax as heading definitions in the configuration file (see "Changing the Sendmail Configuration File" on page 7-35).

M This line contains a message printed by the **mailq** command. It is mainly used to store status information.

P This line contains the priority of the current message. The priority is used to order the queue. Higher numbers mean lower priorities. The priority increases as the message sits in the queue. The initial priority depends on the message class and the size of the message.

R This line contains a receiver address. There is one line for each receiver.

S This line contains the sender address. There is only be one of these lines.

T This line contains the job creation time that is used to compute when to time out the job.

As an example, the following is a **q** file sent to amy@zeus:

```
P217031
T566755281
DdfAA00269
MDeferred: Connection timed out during user open with zeus
Sgeo
Ramy@zeus
H?P?return-path: <geo>
Hreceived: by george (0.13 (NL support)/0.01)
        id AA00269; Thu, 17 Dec 87 10:01:21 CST
H?D?date: Thu, 17 Dec 87 10:01:21 CST
H?F?From: geo
H?M?message-id: <8712171601.AA00269@george>
HTo: amy@zeus
Hsubject: test
```

This example shows the

1. The priority of the message (P217031)
2. The submission time in seconds (T566755281)
3. The name of the data file (DdfAA00269)
4. A status message:

   MDeferred: Connection timed out during user open with zeus
5. The ID of the sender (Sgeo)
6. The ID of the receiver (Ramy@zeus)
7. Heading information for the message (lines beginning with H).

Moving the Queue

In some cases, you may find that a major host going down for a couple of days may create a prohibitively large queue. This will result in **sendmail** spending a large amount of time sorting the queue. This situation can be fixed by moving the queue to a temporary place and creating a new queue. The old queue can be run later when the host returns to service. Use the following procedure to recover from this situation:

1. Use the **ps -e** command to determine the process ID of the **sendmail** daemon, and then use the **kill** command to kill the existing daemon.
2. Move the entire queue directory using the following commands:

   ```
   cd /usr/spool
   mv mqueue omqueue
   ```

3. Start a new daemon:

   ```
   sh /etc/rc.sendmail
   ```

4. When the other system returns to operation, use the following command to run the old mail queue:

   ```
   /usr/lib/sendmail -oQ/usr/spool/omqueue -q
   ```

   The **-oQ** flag specifies an alternate queue directory and the **-q** flag specifies to run every job in the queue. To get a report about progress of the operation use the **-v** flag. This operation can take a long time.

When the queue is empty and you have processed the old log files according to your local procedures (saving them to another directory or to diskette), remove the log files and the temporary directory by entering the following commands:

```
rm /usr/spool/omqueue/*
rmdir /usr/spool/omqueue
```

Flushing the Mail Queue

The mail queue contains a time value that represents the time of submission, rather than the amount of time left until timeout. As a result, you can flush messages that have been in the queue for a short period by running the queue with a short message timeout. For example, the following command runs the queue, flushes anything that is at least one day old (-oT1d), and tries to send all messages in the queue that have not exceeded the timeout value:

```
/usr/lib/sendmail -oT1d -q
```

Changing the Sendmail Configuration File

This section describes the configuration file **sendmail.cf**. Examples that appear in this section are taken from the default configuration file, **/usr/adm/sendmail/sendmail.cf**, that is installed with the **sendmail** program. The syntax of the configuration file allows the program to parse it easily, since every time **sendmail** starts it reads the processed form of the configuration file.

The configuration file consists of a series of control lines, each of which begins with a single character that defines how the rest of the line is used. Lines beginning with a space or a tab are continuation lines. Blank lines and lines beginning with a # are comments. The control line can be used for the following functions:

- Defining macros and classes for use within the configuration file
- Setting options used by **sendmail**
- Defining mail delivery information
- Defining how **sendmail** interprets address information.

The following sections explain how each of these functions are implemented in the configuration file.

Editing sendmail.cf with the edconfig Command

The **/usr/lib/edconfig** command provides a menu interface to defining some of the parameters in the **sendmail** configuration file as listed in Figure 7-7 on page 7-36. In addition, the program provides descriptive text to help you enter the correct information. Use this program to edit the local configuration file or to set up a configuration file (using any file name) to be used on another system. The configuration file must be in the format of the configuration file provided with the **sendmail** program. You must be a member of system group to edit **/usr/adm/sendmail/sendmail.cf** with this or any other program. Use the following procedure to edit the configuration file with this program.

Using edconfig

1. Change to the **/usr/adm/sendmail** directory.

   ```
   cd /usr/adm/sendmail
   ```

2. Start **edconfig** with the configuration file:

   ```
   /usr/lib/edconfig sendmail.cf
   ```

   The **edconfig** program starts and displays a menu.

3. Follow the instructions on the screen to find and change the desired configuration parameters.

4. When you have finished, exit the program by entering the number for writing the configuration file and exiting. If you do not want to keep the changes that you made, enter the number for exiting without writing the configuration file.

Symbol	Meaning	More Information
Dw	Host name macro	"Changing the Host Name Macro" on page 7-37
Cw	Host name class	"Changing the Host Name Class" on page 7-38
DD	The *domain name* macro	"Changing the Domain Name Macro" on page 7-39
DE	A macro that defines the first part of the domain name	"Changing the Domain Name Part Macros" on page 7-39
DF	A macro that defines the second part of the domain name	"Changing the Domain Name Part Macros" on page 7-39
DG	A macro that defines the third part of the domain name	"Changing the Domain Name Part Macros" on page 7-39
DH	A macro that defines the fourth part of the domain name	"Changing the Domain Name Part Macros" on page 7-39

Figure 7-7 (Part 1 of 2). Configuration Parameters that You can Change with edconfig

Symbol	Meaning	More Information
OL	Operational logging level	"Changing the Operational Logging Level" on page 7-41
Od	Default delivery mode	"Setting Delivery Mode" on page 7-40
OA	Alias file path	"Setting Configuration Options" on page 7-40
OS	Statistics file path	"Setting Configuration Options" on page 7-40
OQ	Queue file path	"Setting Configuration Options" on page 7-40
OT	Maximum mail retention time in queue (queue timeout)	"Setting Configuration Options" on page 7-40
Oc	Queue use of expensive mailers	"Setting Configuration Options" on page 7-40
OS	Define path to **mail** statistics file	"Setting Configuration Options" on page 7-40
DZ	Configuration file revision level	"Changing the Configuration File Revision Level" on page 7-42

Figure 7-7 (Part 2 of 2). Configuration Parameters that You can Change with edconfig

Changing the Host Name Macro

Note: As a general guideline to avoid confusion in names for your system, the following names should all be the same:

nodename Defined in **/etc/master** and built into the kernel. Changed with **chparm nodename**=*node*, and displayed with **uname -n**

nickname The name that Distributed Services uses to refer to your system. Defined by the **ndtable** command.

hostname The name that Interface Program for use with TCP/IP uses to refer to your system - defined and displayed by the **hostname** command (usually run from **/etc/rc.tcpip**).

The host name macro, **w**, specifies the name of your host system that is used in the return address of all messages that you generate. Define this macro to be the name of your system returned by the **uname -n** command. Use the following procedure to define that macro.

1. Determine your host name with the **uname -n** command. For example:

```
uname -n
george
_
```

2. Start the **edconfig** command ("Editing sendmail.cf with the edconfig Command" on page 7-35) and edit the host name macro to match the value returned by **uname**. Use the following format:

   Dw*hostname*

   In this format, *hostname* is the name returned from the **uname** command. For example, the following entry defines the host name for system `george`.

```
Dwgeorge
```

Changing the Host Name Class

The host name class, **Cw**, specifies the name and all aliases for your host system. If your system uses different names for two different network connections, enter both names as part of the host name class. If you do not define both names, mail sent to the undefined name is returned to the sender. Define this class to be the name of your system returned by the **uname -n** command and a list of other aliases that your system uses. Use the following procedure to define that macro.

1. Determine your host name with the **uname -n** command. For example:

```
uname -n
george
_
```

2. Determine any aliases that your system uses.

3. Start the **edconfig** command ("Editing sendmail.cf with the edconfig Command" on page 7-35) and edit the host name class to match the value returned by **uname**. Use the following format:

   Cw*hostname alias1 ... aliasn*

   In this format, *hostname* is the name returned from the **uname** command and *alias1* through *aliasn* are alternate names by which your system is known. For example, the following entry defines the host name class for system `george`.

```
Cwgeorge local
```

Changing the Domain Name Macro

The domain name macro, **DD**, specifies the full domain name of your local domain. The domain name is a series of names separated by periods (.) in the following format:

name1.name2.name3.name4

You do not need to have all of the parts in your domain name, but you must specify them if they exist. For example, the following are all properly formatted domain name macro definitions:

```
DDpub.aus.ibm.com
```
or
```
DDrch.ibm.com
```
or
```
DDacct.xyz
```

Refer to *IBM RT Using the AIX Operating System* for a discussion of domain addressing, and to *Interface Program for use with TCP/IP* for information to set up the domain name server for your network.

Changing the Domain Name Part Macros

The domain name part macros specify the individual parts of the full domain name as defined in "Changing the Domain Name Macro." You must define domain name part macros if they exist in the full domain name macro. The values that you enter for each of the domain name part macros must be the same as the corresponding parts that you specify for the full domain name. If you use less than four parts for your full domain name, always begin with the **DE** macro and define as many other macros in order (**DF**, **DG**) as needed. Define the domain name part macros with entries in the following form:

DE*name1*
DF*name2*
DG*name3*
DH*name4*

For example, for a domain name of `pub.aus.ibm.com` enter the following macro definitions:

```
DEpub
DFaus
DGibm
DHcom
```

Setting Configuration Options

You can set configuration options for use by **sendmail** with a control line in the configuration file. The options that can be set are the same as those options that you can specify with the **-o** flag to the **sendmail** command (see *AIX Operating System Commands Reference*). An option is named with a single character. The format of the set option control line is:

O*oVAL*

In this format *o* is a single character name for the option being set and *VAL* is either a string, an integer, a time interval or a boolean option. Legal values for a boolean option are `t` or `T` for a true value and `f` or `F` for a false value. If you do not specify a value for a boolean variable, it becomes a true value.

For example, the following entries from the default **sendmail.cf** file show the format of the set option control line:

`OL9` Sets the log level option variable `L` to a value of 9. This entry occurs early in **sendmail.cf** to ensure that the log is maintained during the reading of the file. See "Logging Mail System Activities" on page 7-23 for more information about the **sendmail** logging facility.

`OQ/usr/spool/mqueue`
Sets the mail queue directory option variable `Q` to a string (`/usr/spool/mqueue`) that defines where the mail log is to be kept.

`OA/usr/adm/sendmail/aliases`
Sets the option variable `A` to the full path name of the aliases file (**/usr/adm/sendmail/aliases**).

Setting Delivery Mode

The **sendmail** program can operate in several delivery modes. The default configuration file sets delivery mode to **b**. However, you can change the delivery mode with the **d** option in the configuration file. These modes specify how quickly mail will be delivered. Legal modes are:

i Deliver interactively (wait until the mail is delivered)

b Deliver in background (run delivery in the background)

q Queue only (let the queue processor deliver the mail).

Mode **i** passes the maximum amount of information to the sender, but is hardly ever necessary. Mode **q** puts the minimum load on your machine, but means that delivery may be delayed for up to the queue interval (see "Determining the Queue Processing Interval" on page 7-29). Mode **b** is probably a good compromise. However, this mode can cause large numbers of processes if you have a mailer that takes a long time to deliver a message.

Setting Time Outs

The **sendmail** program can time out when reading standard input or when reading from a remote SMTP server. The default configuration file sets this value to 5 minutes. This value should be correct for most situations. However, if you need to change the timeout value, change the **r** option in the configuration file.

After sitting in the queue for a few days, a message times out. The **sendmail** program notifies the sender of the message that it could not be sent. The timeout is typically set to three days. Set this timeout with the **T** option in the configuration file.

Changing the Operational Logging Level

The **OL** option specifies the log level to be used when **sendmail** is running. Valid levels and the activities that they represent are (each number includes the activities of all numbers of lesser value and adds the activity that it represents):

Level	Activity Logged
0	No logging
1	Major problems only
2	Message collections and failed deliveries
3	Successful deliveries
4	Messages being deferred (due to a host being down, etc.)
5	Placing messages in the queue (normal event)
6	Unusual but benign incidents (trying to process a locked file, etc.)
9	Log internal queue id to external message id mappings. This can be useful for tracing a message as it travels between several hosts.
12	Several messages that are of interest when debugging.
16	Verbose information regarding the queue.

The default level is `OL9`.

Changing the Configuration File Revision Level

The configuration file revision level macro, **Z**, helps you track changes that you make to the configuration file. Each time that you make a change to the configuration file, you should also change the value of this macro. You can choose any format for the number that you define. For example, if the configuration file is at level `3.1`, the following entry should be in the configuration file:

```
DZ3.1
```

You can also use a text string for this macro:

```
DZversion_one
```

Editing sendmail.cf with a Text Editor

To make changes to **sendmail.cf** other than those that are permitted by the **edconfig** command, you must use a text editor. Most of these parameters should not be changed casually, since they affect how **sendmail** interfaces with the system and networks. Read the following sections carefully to determine whether you need to change any of these parameters. In most cases, you do not need to change them to get your mail system working.

When using an editor on a configuration file that did not come with the **sendmail** installation, be aware that the tab character may be defined (by default) as the field separator character in rule sets when **sendmail** reads the file. Some editors (including the **INed** editor supplied with AIX Operating System) store tabs as the number of spaces they represent instead of the tab character. If you edit and then save such a configuration file with one of these editors and then try to build the configuration file using **/usr/lib/sendmail -bz**, you get a large stream of error messages and the data base version of the configuration file is not built.

Note: The **vi** and **tvi** editors do not change tab characters. This feature allows them to provide both normal and secure editing.

You can solve this problem by changing the field separator character with the **I** option (see "Defining Rewrite Rules" on page 7-63 for other changes that you must do if you elect this way). This change allows you to use your favorite editor without a problem with the field separator. By default, the configuration file that comes with **sendmail** has this option set to define _ (underscore) as the space character, with spaces and tabs accepted as field separators.

Defining Macros and Classes

A ***macro*** is a symbol that represents a value or string. A macro is defined by a **D** control line. A ***class*** is a symbol that represents a set of one or more words. Classes are used in pattern matching when the **sendmail** program is parsing addresses. Macros are not expanded until **sendmail** loads the rule sets when it starts up. The following letters introduce configuration file control lines that define macros and classes to set up the **sendmail** program:

D*xVAL* Defines symbol *x* to be a macro and assigns a value of *VAL* to it. If a second D*x* defines the same symbol, the second definition replaces the first definition.

C*c string* Defines symbol *c* to be a class and assigns a word or group of words (*string*) to it. If a second C*c* defines the same symbol, the *string* from the second definition is added to the *string* from the first definition. No words are deleted from the class definition.

F*c file* Defines symbol *c* to be a class and assigns a word or group of words listed in a separate file to the symbol.

The symbols must be a single character selected from the ASCII set. Use uppercase letters for macros and classes that you define. Lowercase letters and special symbols are macros and classes defined by the program (see "Understanding System-Defined Macros" on page 7-45).

To use a macro or class in a control line, put a $ (dollar sign) before its name. For example, if the name of the macro is x, use $x when using that macro in a control line. Without the preceding $, the program interprets x as only the letter x. To specify conditional expressions, use the format:

$?*x text1* $¦ *text2* $.

In this format, the symbols have the following meaning:

$? If

x The macro being tested (e.g. if *x* is defined)

text1 The pattern to be used if **$***x* is defined.

$¦ Else

text2 The pattern to be used if **$***x* is not defined.

$. Specifies the end of the conditional expression.

The else clause ($¦ *text2*) is not required.

Note: Do not use any of the characters defined as tokens (by the required macro **o**; see "Defining Required Macros" on page 7-47) when defining a word in a class. The **sendmail** program may not be able to read the definition correctly.

Creating a Macro

Use a control line that begins with the letter **D** to define a macro. The syntax for the **D** macro definition is:

D*x* *VAL*

In this format *x* is the name of the macro and *VAL* is the value to assigned to it.

For example, the following control line in the configuration file defines a macro D, representing the name of the local domain, to be the name assigned to that domain. In this example, the local domain is assigned the name 802, but the name could be any alphanumeric string that you or your network administration chooses.

```
DD802
```

The following example shows the use of a previously defined macro to define a new macro. It defines a required macro j, representing the official name for the local site, to be the value returned by the **sendmail** macro $w (the hostname of the local site).

```
Dj$w
```

Creating a Class Using a List

Use a control line that begins with the letter **C** to define a class with a specified list of members. The syntax for the **C** macro definition is:

C*c* *word1* *word2* ...

In this format *c* is the name of the class that matches any of the named words, *word1*, *word2*, etc.. Any word cannot include characters defined as delimiters (such as . (period)). You can spread the assignment of words to the class across many lines. For example, the control line:

```
CDLOCAL 802
```

Defines the class D to represent the class of names: LOCAL and 802. You can accomplish the same function with two separate control lines:

```
CDLOCAL
CD802
```

Creating a Class Using a File

Use a control line that begins with the letter **F** to define a class whose members are listed in an external file. The syntax for the **F** class definition is:

F*c* *file*

In this format *c* is the name of the class that matches any of the words listed in the file *file*.

Understanding System-Defined Macros

The **sendmail** program defines some macros internally for use in writing stanzas in the configuration file. You do not need to define these macros in the configuration file, but you can use the symbols when writing other macros and parsing rules in the configuration file. The following paragraphs describe these macros.

Date Macros

The **sendmail** program defines three macros that represent dates. These macros are:

a The origination date in Arpanet format:

```
Fri, 9 Oct 87 09:25:47 CDT
```

This date is the time extracted from the `Date:` line of the message (if there is one). If no `Date:` line is found in the incoming message, **$a** is set to the current time.

b The current date in Arpanet format:

```
Fri, 9 Oct 87 09:25:47 CDT
```

This macro is used for postmarks.

d The current date in **ctime** format:

```
Fri Oct  9 09:25 CDT 1987
```

Macros Relating to the Sender

The **sendmail** program defines the following macros that identify the sender of the message:

f This macro contains the sender's ID.

g This macro contains the sender's address relative to the receiver.

s This macro contains the name of the sender's host system.

x This macro contains full name of the sender. The **sendmail** program determines this name in one of the following ways:

- It can be passed as a flag to **sendmail**
- It can be the value of the **Full-name:** line in the message heading
- It can be the comment field of a **From:** line
- If the message originated locally, the full name can be looked up in **/etc/passwd**.

Macros Relating to the Receiver

The **sendmail** program defines the following macros that identify the receiver of the message:

h This macro contains the name of the receiving host as set by the **$@** part of the mailer resolution line in rule set 0.

u This macro contains the ID of receiving user as set by the **$:** part of the mailer resolution line in rule set 0.

z If the message is a local message, this macro contains the home directory of the receiving user.

Macros Relating to Message Routing

The **sendmail** program defines the following macros to help it track and route messages through the network:

c This macro contains the hop count. The ***hop count*** is the number of times that this message has been processed and usually corresponds to the number of separate timestamps in the message (some mail handler programs allow you to change the hop count).

i This macro contains the mail queue ID of the message on this host. This macro is useful for tracking messages when put into the timestamp line of the message.

p This macro contains the process ID of the **sendmail** program. The **sendmail** program uses this macro, along with the **$t** macro to create unique strings for the `Message-ID:` field in the message.

r This macro contains the protocol used to communicate with **sendmail**. This macro is not supported.

t This macro contains a numeric representation of the current time. The **sendmail** program uses this macro, along with the **$p** macro to create unique strings for the `Message-ID:` field in the message.

v This macro contains the version name and number of the AIX Operating System running on the system.

w This macro contains the name of the local host.

Defining Required Macros

The configuration file that comes with **sendmail** defines the following macros to transmit information to **sendmail**. If you create a new configuration file, be sure to include definitions for these macros:

`e`	The SMTP entry message
`j`	The official domain name for this site
`l`	The format of the `From` line (not the `From:` line)
`n`	The name of the daemon (for error messages)
`o`	The set of operators in addresses
`q`	The default format of sender address

The following paragraphs describe the use and definition of these macros.

Defining the e Macro

The **e** macro defines the message that the **sendmail** SMTP daemon prints out when another system connects to it. The first word must be the **$j** macro. The definition for this macro in the default configuration file is:

```
De$j Sendmail $v/$Z ready at $b
```

This control line defines the required macro `e` to be the following string:

- The official domain name for the site (defined by the **$j** macro)
- The word `Sendmail`
- The version number of AIX Operating System (defined by the **$v** macro)
- A separating / (slash) character
- The version number of **sendmail.cf** (defined by the **$Z** macro)
- The words `ready at`
- The current date (defined by the **$b** macro).

Defining the j Macro

The required macro **j** defines the official name for the local site. It should be in the same format as all system addresses:

host.domain

In this format, *host* is the name of the local system and *domain* is the hierarchical domain address. Since this information is already defined in the system, the default configuration file defines the **j** macro to be the value returned by the **sendmail** macro **$w** (the hostname of the local site).

```
Dj$w
```

Defining the l Macro

Note:

1. Some mailers (for example, **bellmail**) include a `From` line that they generate, ignoring the **sendmail** `From` line. In that case, changing the **l** macro has no effect on the message heading format.
2. Many mail handler programs use the `From` line as a message delimiter when scanning mailboxes. Changing the format of this line may cause problems for these programs.
3. The **sendmail** `From` line is used when **sendmail** writes mail directly to a file, such as **dead.letter** or a file defined by an alias.

The required macro **l** defines the format for the `From` line in the message heading. In the default configuration file, this definition appears as follows:

```
DlFrom $g  $d
```

This example uses two **sendmail** macros: `$g` is the address of the sender relative to the receiver; `$d` is the date in **ctime** format. This entry results in a line like the following line appearing at the beginning of the message:

```
From root Tue Nov  3 14:24:05 CST 1987
```

Defining the n Macro

The required macro **n** defines the name of the user that will receive error messages pertaining to mail system errors. In the default configuration file, this definition appears as follows:

```
DnMAILER-DAEMON
```

Defining the o Macro

The required macro **o** defines a list of characters that are parsed as separate tokens when reading rules in the configuration file. For example, if **r** were in the **$o** macro, then the input, **address**, would be scanned as three tokens: **add**, **r** and **ess**. In the default configuration file, the definition of this macro appears as follows:

```
Do.:%@!^=/[]
```

In addition, the following characters are *always* recognized by **sendmail** as tokens:

- Ctrl-A
- (
-)
- <
- >
- , (comma)
- ; (semicolon)
- \ (backslash)
- " (quote)
- Carriage return
- Newline

Network organizations, such as ARPA, have standards concerning tokens. Be sure to check with your network organization before changing this macro.

Defining the q Macro

The required macro **q** defines the format of an address in a message if no other format is specified. In the default configuration file, this macro definition appears as follows:

```
Dq$g$?x ($x)$.
```

This entry defines the address to be the address of the sender relative to the receiver (`$g`), and if the **sendmail** variable containing the full name of the sender is defined (`$?x`) that variable is printed within parentheses following the sender address (($x)). The symbol `$.` closes the conditional expression introduced by the symbol `$?`. This format could result in the following addresses:

```
amy@athena.802
amy@athena.802 (Amy Smith)
```

In the second example, the full name of Amy Smith is in **/etc/passwd** on system athena. In the first example, the full name field for user ID `amy` in **/etc/passwd** is empty.

Defining Message Precedence

The configuration file contains lines to define mail queue precedence for messages that contain a `Precedence:` field. Normally, you do not need to change the values in the default configuration file. The name defined and the numerical value assigned are based on the needs of the network. Higher numbers have higher priority; numbers less than zero indicate that error messages will not be returned to the sender of these messages. Precedence value is zero for any precedence name not defined in this file. For example, the configuration file may contain the following entries:

```
Pfirst-class=0
Pspecial-delivery=100
Pbulk=-60
Pjunk=-100
```

These entries set `special-delivery` as the highest priority message and `junk` as the lowest priority.

Defining Administrative IDs

Administrative IDs are those IDs that can override the sender address using the **-f** flag to the **sendmail** command. The configuration file defines these IDs with the **T** control line. For example, the configuration file may contain the following entries:

```
Troot
Tdaemon
Tuucp
```

These entries define IDs `root`, `daemon` and `uucp` as administrative IDs for the **sendmail** command.

These IDs could have been defined using only one **T** control line:

```
Troot daemon uucp network
```

Defining Message Headings

The **sendmail** program expects mail to have the following parts in order:

1. An operating system `From ` line (defined by the five characters: F, r, o, m and space)
2. Mail header lines that begin with a keyword followed by a colon, such as `From:` or `To:`
3. An empty line
4. The body of the message.

The **sendmail** program detects the operating system `From` line by checking the first five characters of the first line. After that, header lines are processed. When it detects a line that does not begin with a keyword followed by a colon, it ends header line processing. If an empty line occurs at that point, it is ignored.

Mailer flags or the mailer determine if an operating system `From` line is generated. Other header lines are present (or not) depending upon those defined in the **sendmail** configuration file, those specified by mailer flags, and those present in incoming mail.

Note: The **bellmail** program generates a `From` line on all local deliveries. The **sendmail** mailer flags do not allow you to alter this.

Lines in the configuration file that begin with a capital letter **H** define the format of the headers used in messages. The format of the **H** line is:

`H[?`*mflags*`?]`*fieldname*`:` *content*

In this format, the variable parameters have the following meaning:

mflags — This is an optional field. If you supply it, surround it with ? (question marks). This field contains mailer flags (see "Specifying Mailer Flags" on page 7-53) that determine whether this H line is used. If the mailer being used requires the information specified by the mailer flag, then this H line is included when formatting the heading. Otherwise, this H line is ignored.

fieldname — This field contains the text that is displayed as the name of the field in the heading information. The actual text used is a matter of choice. Some typical field names include: From, To, and Rcvd From.

content — This field defines the information that is displayed following the field name. It usually uses a **sendmail** macro to specify the information.

The following example lines are from a typical **sendmail.cf** file:

```
H?P?Return-Path: <$g>
```

This line defines a field called `Return-Path` that displays the content of the **$g** macro (sender address relative to the receiver). The `?P?` portion indicates that this line is only used if the mailer use the **P** flag (the mailer requires a Return-Path line).

```
HReceived: $?sfrom $s $.by $j ($v/$Z)
        id $i; $b
```

This line defines a field called `Received`. This field displays the following information:

`$?sfrom $s $.`
: If an `s` macro is defined (sender's host name), display the text `from` followed by the content of **$s**.

`by $j` — Display the text `by` followed by the content of **$j** (official name for this site).

($v/$Z) Display the version of the **sendmail** program (**$v**) and the version of **sendmail.cf** (**$Z**), set off by parentheses and separated by a slash.

id $i; Display the text `id` followed by the content of **$i** (mail queue ID of the message) and a **;** (semi-colon).

$b Display the current date.

Defining a Mailer

Note: Defining a mailer entry in the configuration does not ensure that it will be used. You must also define parsing rules (see "Changing the Format of Addresses" on page 7-58) that resolve to that mailer.

A ***mailer*** is a program that delivers mail either locally or over some type of network to another system. Use control lines that begin with the letter **M** to define the characteristics of a mailer program that interfaces with **sendmail**. The format of a mailer definition control line is:

M*mname*, P=*path*, F=*flags*, S=*xx*, R=*yy*,[E=*eol*,] [A=*argv*,] [M=*limit*]

The following paragraphs describe the parameters for the mailer definition. Some examples follow those descriptions.

Specifying a Mailer Name

Each mailer must have an internal name. The name can be any string that you choose, except that the names `local` and `prog` are reserved for the mailers for local delivery and delivery to programs. You must provide definitions for these two mailers in **sendmail.cf** if they are not already there (the default configuration file contains these definitions). To define the mailer name, put the name immediately after the **M** in the mailer definition control line:

M*mname*

For example, the following segment introduces the definition line for a mailer called `lan`:

```
Mlan
```

Defining the Path to the Mailer Program

Specify the location of the mailer program with the **P** field in the mailer definition. This field has the format:

P=*path*

The *path* defines the full path name of the mailer program on the local system. If the mailer program is the **sendmail** version of SMTP, use the string [IPC] as the path. For example, the following two mailer definition fragments define a local mailer at **/bin/bellmail** and another mailer that is the **sendmail** implementation of SMTP.

```
Mlocal, P=/bin/bellmail,
Mlan, P=[IPC],
```

Specifying Mailer Flags

Mailer flags provide further information to **sendmail** about the mailer program being described. Specify mailer flags with the **F** field in the mailer definition. This field has the format:

F=*flags*

Figure 7-8 defines the meaning for the flags that **sendmail** recognizes. For example, the following mailer definition fragment uses the flags rlsm to indicate that the mailer requires a **-r** flag, does local delivery, needs quotes stripped from addresses, and can deliver to more than one user at a time:

```
Mlocal, P=/usr/bin/mail, F=rlsm,
```

Flag	Meaning
C	If this flag is set, this mailer inspects the address of any incoming mail that it processes for the presence of an @ sign. If it finds an @, it saves the @ and the remainder of the address to be used when rewriting addresses in header lines in the message (when mail is forwarded to ANY mailer). The receiving mailer adds the saved portion of the address to any address that does not contain an @ after the address has been processed by rule set 3 (this processing does not depend upon a mailer flag; it always occurs). Do not use this flag for general operation, since it does not interpret complex, route-based addresses properly.
D	This mailer needs a **Date:** or **Resent-Date:** header line.

Figure 7-8 (Part 1 of 3). Mailer Flags

Flag	Meaning
e	This mailer is expensive to connect to. If the **c** configuration option is set, mail for this mailer is always placed in the queue.
E	Escape lines beginning with the exact five characters F, r, o, m, space in the body of the message with a > (greater than symbol). This flag allows operating system `From` lines (or any other text lines beginning with those five characters) to appear in the body of the message without being interpreted as the start of a new message.
f	The mailer needs a **-f** flag. The flag is inserted into the call for the mailer followed by the expansion of the **$g** macro (sender's address relative to the receiver).
F	This mailer needs a **From:** or **Resent-From:** header line.
h	Preserve uppercase letters in host names for this mailer.
I	This mailer uses SMTP to communicate with another SMTP server that is part of the **sendmail** program (not to an SMTP server that is not a part of **sendmail**). When communicating with another **sendmail** program, the mailer can use features that are not part of the standard SMTP protocol. This option is not required, but if this option is omitted the transmission will still operate successfully but not as efficiently as possible.
l	This mailer is local; final delivery will be performed.
L	For SMTP, restricts data transmission and receiving to 7-bit data unless the **N** flag is set. If the **N** flag is set, full 8-bit data transmission is used to support the national language character set. The **L** flag also enforces SMTP line lengths. Lines that are too long are broken up. If the **L** flag is not set, full 8-bit data transmission is used to and from mailer programs.
m	This mailer can be sent to multiple users on the same host in one transaction. When a **$u** macro occurs in the **argv** part of the mailer definition, that field will be repeated as necessary for all qualifying users.
M	This mailer needs a **Message-ID** or **Resent-Message-ID:** header line.

Figure 7-8 (Part 2 of 3). Mailer Flags

Flag	Meaning
n	Do not insert an operating system **From** line on the front of the message.
N	This mailer supports the national language character set. This flag modifies the **L** flag to allow 8-bit data transmission through SMTP.
p	Use the Return-Path in the SMTP **MAIL FROM:** command rather than just the return address; although this type of return address is required for the transfer protocol, many hosts do not process return-paths properly.
P	This mailer needs a **Return-Path:** header.
r	Same as **f**, but sends a **-r** flag. Specify this flag to pass the name of the sender as a **-r** flag under certain circumstances.
s	Strip quote characters off of the address before calling the mailer.
S	Don't reset the user ID before calling the mailer. Use this flag in a secure environment where **sendmail** runs as root to help avoid forged addresses. This flag is suppressed if given from an unsafe environment, for example, a user's **.mailrc** file. If the mailer must be called as **root**, use the **S** flag.
u	Preserve uppercase letters in user names for this mailer.
U	This mailer needs AIX-style **From** lines with the UUCP-style **remote from <host>** on the end.
x	This mailer needs a **Full-Name:** header line.
X	This mailer uses a hidden dot algorithm that adds an extra dot to the beginning of any line that begins with a dot. The leading dot must then be removed at the other end of the transmission. This method ensures that lines in the message that contain a dot do not end the message prematurely.

Figure 7-8 (Part 3 of 3). Mailer Flags

You can also define mailer flags to match flags that you define in special header definitions in your **sendmail** configuration file.

Specifying the Rewrite Rule Set for the Mailer

The **sendmail** program uses sets of rewrite rules to change the format of incoming addresses to a style that the receiving mailer program can understand. These rewrite rules are described in "Changing the Format of Addresses" on page 7-58. Specify the rewrite rule set to use on sender addresses for this mailer with the **S** field; specify the rewrite rule set to use on receiver addresses for this mailer with the **R** field. These fields have the following format:

`S=`*xx*`, R=`*yy*`,`

In this format *xx* and *yy* are integers that specify a particular rule set number for processing the addresses for this mailer. These rules can perform operations such as appending the current domain to addresses that do not already have a domain. For example, if the following header were being processed by the local **sendmail** program, it would require further processing before being sent to another location:

```
From: geo
```

If it were sent on a domain address network, it could be changed to:

```
From: geo@zeus.aus
```

If it were sent on a **uucp** network, it could be changed to:

```
From: zeus!geo
```

Defining a Different End of Line Character

The normal indication of the end of line is a string that contains only the new line character. To change this character, use the optional **E** field. The format of this field is:

`E=`*eol*`,`

In this format, *eol* is the character string that specifies the end of the line. You can use normal \ escape characters to specify the end of line character (\r, \n, \f, \b).

Passing Information to the Mailer

Specify information to be passed to the mailer with the **A** field. This field has the following format:

`A=`*string*

In this format, *string* can be any string of words with imbedded spaces allowed. Any or all of the words can be symbols, such as **$u** (receiving user name) defined in **sendmail.cf**. If you do not include this field, or the field does not contain the **$u** symbol, **sendmail** uses the SMTP protocol to send messages to the mailer. If the **P** field for this mailer (access path name) is [IPC], indicating a mailer accessed with interprocess communications, use the following information in this field:

`A=IPC $h [`*port*`]`

In this notation *port* is the optional port number to which to connect.

Limiting Message Size

Use the **M** mailer flag to specify the maximum size in bytes of messages that the mailer program handles. For example, the following field specifies a maximum limit of 10,000 bytes:

```
M=10000
```

Example Mailer Specifications

The following mailer specification specifies a local delivery mailer:

```
Mlocal, P=/bin/bellmail, F=lsDFMmn, S=10, R=20, A=mail $u
```

It is called **local** and is located in the file **/bin/bellmail**. The mailer takes the following flags:

l	Local delivery
s	Strips quote characters from addresses
DFM	Needs `Date:`, `From:` and `Message ID` fields.
m	Delivers to multiple users
n	Does not need an operating system `From` line at the start of the message.

Rule set 10 should be applied to sender addresses in the message and rule set 20 should be applied to receiver addresses. Additional information sent to the mailer in the **A** field is the word, **mail** and words containing the name of the receiving user. If a **-r** flag is inserted it will be between the words, **mail** and **$u**.

The following mailer specification specifies a mailer for local area network delivery:

```
Mlan, P=[IPC], F=meC, S=11, R=21, A=IPC $h, M=100000
```

It is called **lan** and is connected to via the **sendmail** internal SMTP mailer. It can handle multiple users at once (**m**), the connection is defined as *expensive* (**e**), and any domain from the sender address on incoming mail is saved for use by an outgoing mailer (**C**). Sender addresses should be processed by rule set 11 and receiver addresses by rule set 21. There is a 100,000 byte limit on messages passed through this mailer (**M**).

Changing the Format of Addresses

The **sendmail** program can receive addresses in a number of different formats and output them in a format needed for the network protocol (uucp or tcp/ip) or mailer program being used to route the message. To perform this translation, **sendmail** uses a set of rewrite rules that are defined in **sendmail.cf**. The **sendmail.cf** that is installed with the **sendmail** program contains enough rules to perform the translation for uucp and tcp/ip networks using a domain address structure. You should not have to change these rules unless you are connecting to a system that uses a different addressing scheme. This section describes the operation of the address rewrite rules so that you can change them for special cases.

Mail address processing consists of the following steps:

1. Receive the mail through the incoming mailer. Save headers and text.
2. Define the sender.
3. Define all recipients.
4. Transmit the mail to the outgoing mailer, change headers of messages as they are transmitted.

Define the Sender

The **sendmail** program determines the sender of a message from one of the following sources:

- **sendmail** user ID (mail originator).
- **sendmail** "From" flag argument (from administrative user).
- **sendmail** SMTP interface for incoming mail.
- "From" type header lines inside the mail when **sendmail** is processing in ARPA-FTP mode (**sendmail -ba**).

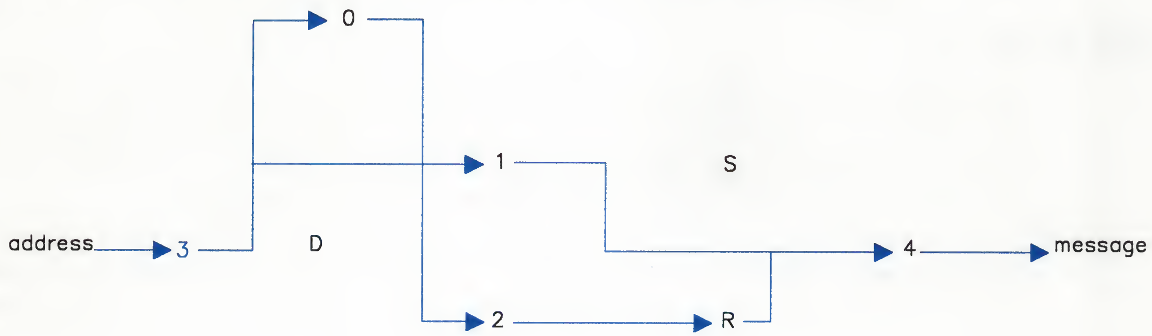
Figure 7-9 shows a flow chart for processing the address of the sender. The following steps occur during processing:

Input	Apply to Input	Resulting Output
From address	Rule set 3, then rule set 0	mailer/host/user (defined by the $#$@$: rule)
user part from rule set 0	Rule set 2, mailer rule set R, then rule set 4	The From address changed to a mail target address
From address	Rule set 3, rule set 1, then rule set 4	Input defined as the **$f** macro

If the mailer has **C** flag - define *from domain* for later usage.

Value of **$f** macro	Search for @ token.	If @ found, save from *domain* as the token string from the first @ to the end of the token string, else save NULL string.

The **$f** macro represents the sender name as seen from the local host.



D = Rules to add the sender domain
S = Rules (for a specific mailer) for processing sender headings
R = Rules (for a specific mailer) for processing receiver headings

A5ACG023

Figure 7-9. Sender Address Rule Set Processing Flowchart

Define the Receiver

The **sendmail** program determines the receiver of a message from one of the following sources:

- **sendmail** mail address parameters.
- SMTP interface for incoming mail.
- If the **-t** flag is used, `To` lines inside mail add to the receiver list. All the following processing takes place for each receiver. Each receiver may have a different mailer associated with it.

Figure 7-10 on page 7-61 shows a flow chart for processing the address of the receiver. The following steps occur during processing:

Input	Apply to Input	Resulting Output
To address	Rule set 3, then rule set 0	mailer/host/user (defined by the $#$@$: rule)
user part from rule set 0	Rule set 2, mailer rule set R, then rule set 4	The `To` address changed to a mail target address

This processing is the same as sender processing with the exception of defining the **$f** macro.

Delivering Mail

The deliver function takes place after the sender and all receivers have been registered. Mail is delivered in separate stages for each separate mailer mentioned in the receiver list. For each mailer, the **$g** macro is defined from the **$f** macro, according to the flowchart for a sender type header described in "Processing Headers" on page 7-61. Then all mail for that mailer is sent. As the mail is sent, header lines are processed.

If mail is sent through **sendmail**'s SMTP interface, then the SMTP `MAIL FROM` command uses the expanded **$g** macro for the address to be presented to the other SMTP program. The SMTP `RCPT TO` command uses the header line processing sequence for `To` type headers to prepare the address for the other daemon.

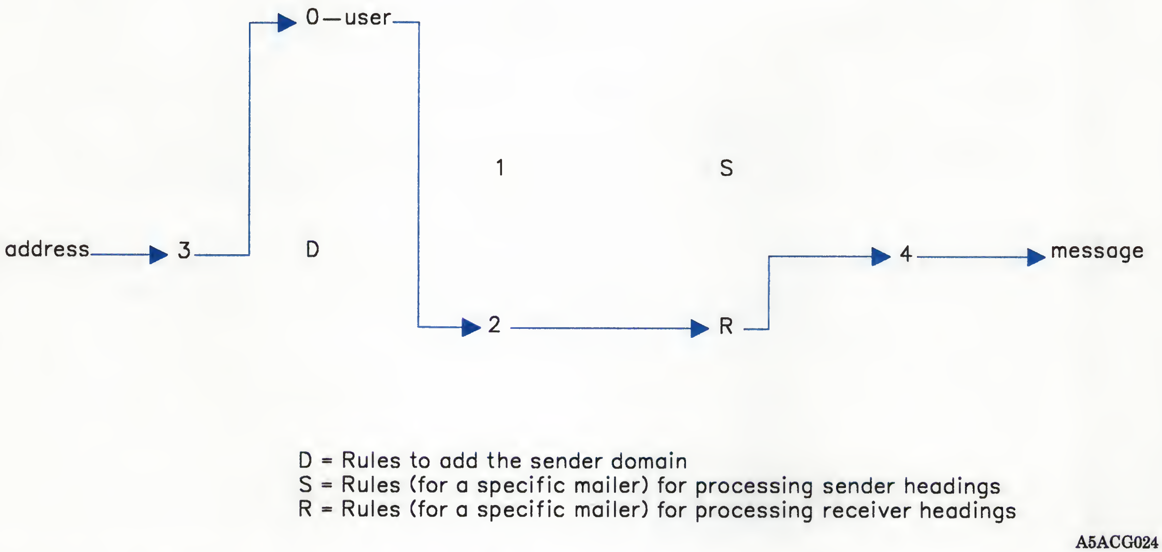


Figure 7-10. Receiver Address Rule Set Processing Flowchart

Processing Headers

As mail is delivered to an outgoing mailer, headers collected from the received mail are sent to the outgoing mailer. (This process can generate new headers, also). As the headers go out, they are changed into a form that is suitable for the next host. Also, after each header is changed, any resulting macro references in the final result are expanded.

Headers may include reference to macros (such as **$g**) that get created using rewrite rules. As the header is processed, segments of the header string may get processed twice by the same rule sets.

This process is repeated for each separate mailer. Figure 7-11 shows the flow chart for processing headers. The following steps occur during processing:

Input	Apply to Input	Resulting Output
To header address	Rule set 3, D processing, rule set 2, mailer rule set R, then rule set 4.	Proper header format for outgoing mail
From header address	Rule set 3, D processing, rule set 1, mailer rule set S, then rule set 4	Proper header format for outgoing mail

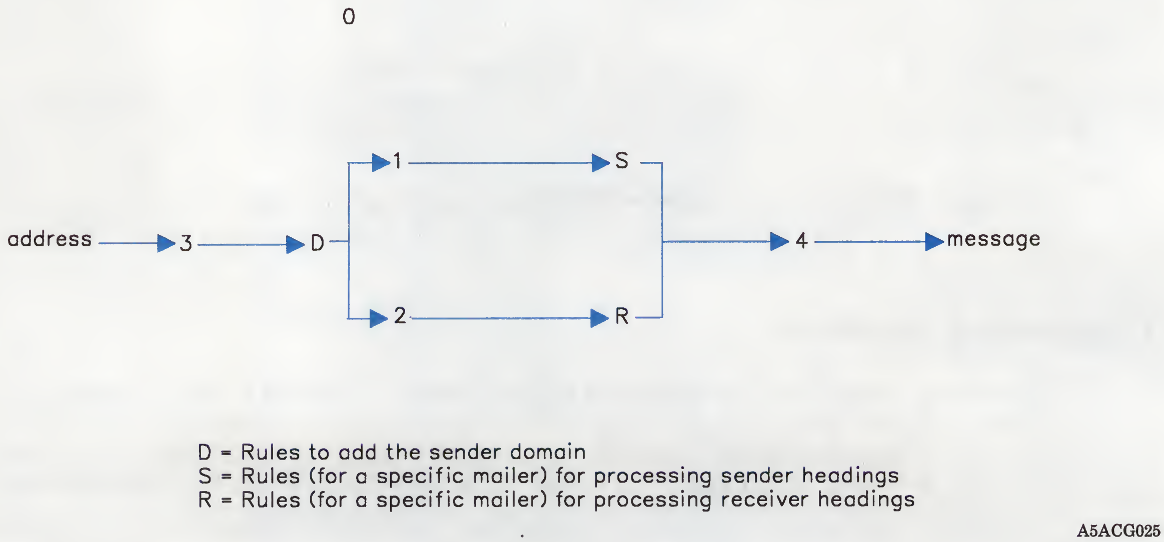


Figure 7-11. Header Rule Set Processing Flowchart

For automatic appending of sender domain, all the following must be true:

- The receiving mailer at this host must have been configured with the **C** mailer flag.
- There must have been a domain (@ token) present in the sender address of a message received through that mailer.
- A receiver for that message must not have any domain at all (no @ token). This takes place on a per receiver basis, since any message may have more than one receiver.

The test for the presence of a domain in the receiver address is to check for an @ token. The header processing is the only processing where automatic appending of sender domain can occur.

Defining Rewrite Rules

Use control lines that begin with the letter **R** to define rewrite rules in the configuration file. The format of a rewrite rule control line is:

R *left_side* <*sep*> *right_side* <*sep*> *comments*

In this format, the notation <*sep*> indicates a string of one or more separator characters. The separator character is determined by the **I** configuration option. If the **I** option is not set, the separator character is the tab character. If the **I** option is set:

- The separator character is the tab or space character
- Spaces that occur inside the *left_side* and *right_side* are represented by a different character.

The different character used to represent a space is defined by the **I** option. In the default configuration file, the **I** option specifies the _ (underscore) to take the place of spaces in rules. After **sendmail** separates the *left_side* from the *right_side*, it changes the special **I** option characters in the left and right sides to spaces.

The *left_side* contains a set of characters that define an incoming address format. The *right_side* contains a set of characters that define the format to which the incoming address is to be changed. The **sendmail** program ignores the comment field. The following example is a rule from the **sendmail.cf** file:

```
R$+!$+          $:$2<@$1.UUCP>          resolve uucp names
```

The meanings of the symbols are explained in "Defining an Input Pattern" on page 7-64 and "Defining an Output Pattern" on page 7-65. The parts of this line are:

R
: The letter R designates that this is a rule line.

$+!$+
: This set of characters is the left side of the rule. It defines the pattern to match against an address.

$:$2<@$1.UUCP>
: This set of characters is the right side of the rule. It defines the pattern to replace the address matched by the left side.

resolve uucp names
: This phrase is the comment of the rule. It is ignored by **sendmail** and may explain the purpose of the rule.

When **sendmail** processes an address it tries to match the pattern of that address format to a pattern specified by the left side of one of the rewrite rules in **sendmail.cf**. When it

finds a rule that matches, it changes the format of the incoming address to the format specified by the right side of that rewrite rule.

Defining an Input Pattern

The left side of a rewrite rule defines an pattern to match against the address that **sendmail** is currently examining. The pattern uses a combination of literal characters, words and special symbols. All special symbols are two characters that begin with a $ (dollar sign). These special symbols are:

`$*`	Match zero or more tokens
`$+`	Match one or more tokens
`$-`	Match exactly one token
`$=`*x*	Match any token in class *x*
`$~`*x*	Match any token not in class *x*

A ***token*** is a set of one or more characters that **sendmail** treats as a single unit when rewriting the address. For example, if the character `@` (at sign) has been defined as the field delimiter in an address, then in the address, `amy@zeus`, **sendmail** recognizes three tokens: `amy`, `@` and `zeus`.

If the input address matches the left side of a rule, the token sequence that matches each special symbol is saved in a numbered special symbol `$`*n*. In this format *n* is an integer that indicates the position of the special symbol in the left side (counting from left to right). Each of the numbered special symbols is passed to the right side of the rule for reformatting. For example, the left side of a rule may be the following expression:

```
$-@$+
```

This rule can be interpreted as:

1. Match any one token and assign it to the symbol **$1**
2. Look for a literal `@` as the next token.
3. Match one or more tokens following the `@` and assign them to the symbol **$2**.

Apply this rule to the following address:

```
amy@zeus.aus.IBM.COM
```

This address contains six tokens: amy, @, zeus, aus, IBM and COM. The periods delimit the tokens in the domain name. The rule matches the input exactly. The values passed to the right side of the rule are:

$1	`amy`
$2	`zeus.aus.IBM.COM`

This rule receives a domain-format address and separates the user ID from the domain address.

Defining an Output Pattern

When the left side of the rule matches the input, the input that matched special characters is saved in the special symbols of the form $*n*, the rest of the input is deleted, and the address is built again using the instructions in the right side.

The right side of a rewrite rule defines a pattern to create a new address from the saved portions of the input address. The pattern uses a combination of literal characters, words and special symbols. All special symbols begin with a $ (dollar sign). These special symbols are:

$*n* Substitute saved token *n* from left side.

$[*name*$] Get an address in standard format for *name*. The **sendmail** program uses the **gethostbyname** subroutine to resolve a possible alias to the official name of the host. If the address is an internet protocol address, it uses the **gethostbyaddr** subroutine to try to convert this address to a domain name, if possible.

$>*n* Call rule set *n*. This symbol substitutes the remainder of the address as indicated by the remaining symbols, and then passes it as the address to be examined by rule set *n*. The final value of rule set *n* then becomes the output address for this rule.

$#*mailer*$@*host*$:*user*
Use this complex symbol only in rule set 0 (see "Changing the Format of Addresses" on page 7-58). It stops evaluation of the rule set and indicates to **sendmail** that the address is resolved to a mailer. This special symbol specifies the mailer, host, and user necessary to direct the mailer. If the mailer is local, the host may be omitted. The *mailer* and *host* must each be a single word, but *user* may contain more than one word.

$@ This symbol causes the rule set to return with a value that is the remainder of the right side of this rule.

$: This symbol terminates the rule, but allows the rule set to continue. Use this symbol to avoid continued application of a rule. For example, the following rule matches anything, passes that input to rule set 7 and continues:

```
R$+                 $:$>7$1
```

The `$:` prevents the rule from being evaluated again to avoid an infinite loop.

When **sendmail** evaluates the right side of a rule it uses the following order:

1. Substitutes parameters from the left side
2. Resolves host names
3. Calls subroutines
4. Processes the symbols `$#`, `$@` and `$:`.

Naming a Set of Rewrite Rules

The configuration file contains several sets of rewrite rules. The **sendmail** program uses some of the sets internally; the format of these sets cannot change. Other sets whose format can be changed can be referenced by mailer definitions and other sets of rewrite rules. Each set of rules is delimited by a control line that contains the letter **S** followed by a number:

```
Sn
```

This control line means that all rewrite rules that follow this control line, until another **S** control line occurs, are part of rule set number *n*. The variable *n* can be any integer. For example, the following statement marks the beginning of rule set 10:

```
S10
```

If you start the same rule set more than once, only the rules contained in the last rule set are kept. In the following generalized example only rules x, y and z are part of the final rule set 11:

```
S11
   rule a
   rule b
   rule c
S11
   rule x
   rule y
   rule z
```

Testing the Rewrite Rules

When you build a configuration file, you can test the rewrite rules using the test mode of **sendmail**. To use the test mode on a new configuration file, first build the data base version of the configuration file and then use the test mode of the **sendmail** command:

```
/usr/lib/sendmail -bz -Cnewfile
/usr/lib/sendmail -bt -Cnewfile
```

In this format, *newfile* is the name of the new configuration file to test. For example, to test a new configuration file named `test.cf`, use the commands:

```
/usr/lib/sendmail -bz -Ctest.cf
/usr/lib/sendmail -bt -Ctest.cf
```

This command reads the configuration file **test.cf** and enters test mode. In test mode, you enter lines of the form:

ruleset address

where *ruleset* is the number of the rewrite rule set to use and *address* is an input address. Test mode shows you the steps it takes and the final address. To test more than one rule

set, use a list of rule set numbers with commas between the rule set numbers. The rules will be applied to the input address in the order that they appear in the list.

For example, the following entry first applies ruleset three to the input, `zeus:geo`. Ruleset one is then applied to the output of ruleset three, followed similarly by rulesets twenty-one and four.

```
3,1,21,4 zeus:geo
```

Using the standard configuration file **/usr/adm/sendmail/sendmail.cf**, the following sequence occurs:

```
#/usr/lib/sendmail -bt -C/usr/adm/sendmail/sendmail.cf
ADDRESS TEST MODE
Enter <ruleset> <address>
> 3,1,21,4 zeus:geo
rewrite: ruleset  3   input: "zeus" ":" "geo"
rewrite: ruleset  6   input: "geo" "<" "@" "zeus" ">"
rewrite: ruleset  6 returns: "geo" "<" "@" "zeus" ">"
rewrite: ruleset  3 returns: "geo" "<" "@" "zeus" ">"
rewrite: ruleset  1   input: "geo" "<" "@" "zeus" ">"
rewrite: ruleset  1 returns: "geo" "<" "@" "zeus" ">"
rewrite: ruleset 21   input: "geo" "<" "@" "zeus" ">"
rewrite: ruleset 21 returns: "geo" "<" "@" "zeus" ">"
rewrite: ruleset  4   input: "geo" "<" "@" "zeus" ">"
rewrite: ruleset  4 returns: "geo" "@" "zeus"
>
```

Use the End of File character (**Ctrl-D**) to exit the program.

For more detail, use the **d21** flag to turn on more debugging (you may find level **d21.12** useful). For example, the following command turns on a large amount of information; a single word address is probably going to print out several pages of information.

```
sendmail -bt -d21.99
```

For example, when using the previous address as an example, a portion of the rule set 4 information that pertains to the rule:

```
R$*<$+>$*                    $1$2$3
```

appears as follows:

```
   .
   .
   .
rewrite: ruleset  4   input: "geo" "<" "@" "zeus" ">"
   .
   .
   .
-----trying rule: "^P" "<" "^Q" ">" "^P"
ap="geo", rp="^P"
ap="geo", rp="<"
ap="<", rp="<"
ap="@", rp="^Q"
ap="zeus", rp=">"
ap=">", rp=">"
ap=<null>, rp="^P"
-----rule matches: "^U1" "^U2" "^U3"
$1: 3fffe05b="geo"
$2: 20011397="@" 3fffe054="zeus"
$3:
rewritten as: "geo" "@" "zeus"
   .
   .
   .
```

Chapter 8. Managing Asynchronous Terminal Emulation (ATE)

CONTENTS

About This Chapter

This section describes how to customize and configure the Asynchronous Terminal Emulation (ATE) program to fit your particular needs. Topics covered include modifying local settings, altering connection settings, remapping control keys, and changing the default file. This chapter also discusses two file transfer protocols: xmodem and pacing.

Before You Begin

Unless a connection has been made to a remote terminal, you must start the ATE program.

The commands in this chapter can be given from either the Unconnected or the Connected Main Menu.

Modifying Local Settings (modify)

The **modify** command lets you change several features of your local system. You can:

- Change the name of the capture file that receives incoming data.
- Switch or toggle the following features *on* or *off*.
  - Add a linefeed character at the end of each line of incoming data.
  - Set echo mode.
  - Emulate a DEC VT100 terminal at the console.[1]
  - Write incoming data to a capture file as well as to the display.
  - Use an Xon/Xoff (transmitter on/off) signal.

If you want to see the Modify Menu to view the features that can be changed and see the current default settings, type the letter **m** after the command prompt on either main menu.

---

1 DEC and VT100 are trademarks of Digital Equipment Corporation.

The Modify Menu with the default values that are in effect each time you start ATE looks like this:

```
Node: Evelyn           MODIFY LOCAL SETTINGS

COMMAND       DESCRIPTION              CURRENT  POSSIBLE CHOICES

Name          Name the capture file    kapture  Any valid file name

Linefeeds     Linefeeds added          OFF      ON, OFF
Echo          Echo mode                OFF      ON, OFF
VT100         VT100 emulation          OFF      ON, OFF
Write         Write display data to    OFF      ON, OFF
                capture file
Xon/Xoff      Xon/Xoff signals         OFF      ON, OFF

To change a value, type the first letter of the command, and press Enter

>
```

A5ACG017

The features that can be changed are in the first or Command column. For a description of each feature, see "Description of Features" on page 8-6. Remember that:

- You can change as many features as you want at the same time.
- If you change the **name** variable, you also must supply a value, the new name.
- All other variables are switches that you can turn `on` or `off` by typing the command. When you type the command, you switch or toggle the value.

You can invoke the **modify** command from either the Unconnected Main Menu or the Connected Main Menu. If you follow the steps in the instruction box below, you can bypass the Modify Menu.

Modifying Local Settings from a Main Menu

1. On the Main Menu, type **m** *commandinitial* [*value*] (*commandinitial*) after the command prompt.

   For example, to toggle the settings of both the **linefeed** and **echo** features, type:

   ```
   >m l e
   ```

   To change the **name** variable to `schedule` in addition to toggling the **linefeed** and **echo** settings, type:

   ```
   >m n schedule l e
   ```

2. Press the **Enter** key.

   Any data you save now is put into the `schedule` file, and the **linefeed** and **echo** commands are switched to the alternate setting.

Additional Information

- The **( )** (parentheses) indicate that you may repeat the contents as many times as you want.
- The [ ] (brackets) indicate that a value is optional. A value applies only to the **name** variable, since all the other variables are `on/off` switches.
- If you give the **modify** command without typing a variable initial, the system displays the Modify Menu. You then must type the initial of each variable you want to change after the command prompt.
- If you use a value with any feature other than **name**, an error message appears:

  ```
  062-003  The 'command-name' command is not valid.
           Enter the first letter of a command
           from the list on the menu.
  ```

  If you see this message, either you have typed an incorrect letter or included an invalid value.

Description of Features

name File for incoming data when the **write** command is `on`, or when **Ctrl-B** is used during a connection.

Options: Any valid file name that is 40 characters or less. The first 18 characters are displayed in the Modify Menu.

Default: *kapture*.

linefeeds Adds a linefeed character after every carriage return in the incoming data stream.

Options: On, Off.

Default: Off

echo Displays your typed input.

With a remote computer that supports echoing, each character you send returns and appears on your display. When **echo** is `on`, each character displays twice, as you type it and when it returns. When **echo** is `off`, each character displays only once, when returned from the remote computer.

Options: On, Off.

Default: Off.

VT100 The local console emulates a DEC VT100 terminal so you can use DEC VT100 codes with the remote system. With **VT100** `off`, the local console functions like an IBM RT.

Options: On, Off.

Default: Off.

Note: Use this command only if you have the keyboard that came with your system, since some keys are remapped. Any other keyboard gives you unpredictable results. Some DEC VT100 codes, like 132 columns, double-height and -width lines, origin mode, and scrolling regions, are not supported.

write Routes incoming data to the file specified in the **name** command, as well as to the display. This functions like the **Ctrl-B** control key during a connection. Carriage return/linefeed combinations are converted to linefeeds before being written to the capture file. In an existing file, data is appended to the bottom.

Options: On, Off.

Default: Off

Xon/Xoff Controls data transmission at a port as follows:

When an Xoff is received, transmission stops.

When an Xon is received, transmission resumes.

An Xoff is sent when the receive buffer is nearly full.

An Xon is sent when the buffer is no longer full.

Options: On, Off.

Default: Off

Altering Connection Settings (alter)

The **alter** command lets you change several data transmission characteristics, including:

- Bit length and rate
- Stop and parity bits
- Port name
- Modem dialing prefixes and suffixes
- Waiting time and retry limits
- File transfer method
- Pacing character or delay time.

If you want to see the Alter Menu to view the characteristics that can be changed and the current default settings, type the letter **a** after the command prompt on either main menu.

The Alter Menu with the original default values in effect each time you start ATE looks like this:

```
Node: Evelyn           ALTER CONNECTION SETTINGS

 COMMAND       DESCRIPTION               CURRENT  POSSIBLE CHOICES

Length        Bits per character        8        7,8
Stop          Number of stop bits       1        1,2
Parity        Parity setting            0        0=none, 1=odd, 2=even
Rate          Number of bits/second     1200     50,75,110,134,150,300,600
                                                   1200,1800,2400,4800,9600,19200
Device        /dev name of port         tty0     tty0-tty16
Initial       Modem dialing prefix      ATDT     ATDT, ATDP, etc.
Final         Modem dialing suffix               0 for none, valid modem suffix
Wait          Wait between redialing    0        seconds between tries
Attempts      Maximum redial tries      0        0 for none, a positive integer
Transfer      File transfer method      p        p=pacing, x=xmodem
Character     Pacing char or number     0        0 for none, a single char/integer

To change a current choice, type the first letter of the command
followed by your new choice (example: r 300) and press Enter.
>
```

A5ACG017

The characteristics that can be changed are in the first or Command column. For a description of each characteristic, see "Description of Characteristics" on page 8-9. Remember that:

- You can change as many features as you want at a time.
- You must type a new value for each characteristic you change. For example, if you want to change the number of bits per second (the rate) to 9600, type:

  ```
  >r 9600
  ```

You can invoke the **alter** command from either the Unconnected Main Menu or the Connected Main Menu. If you follow the steps in the instruction box below, you can bypass the Alter Menu.

Altering Connection Settings from a Main Menu

1. On the Main Menu, type **a** *commandinitial value* (*commandinitial value*) after the command prompt. For example, to change the **rate**, **wait**, and **attempt** values, type:

   ```
   >a r 9600 w 5 a 1
   ```

2. Press the **Enter** key.

Additional Information

- The **( )** (parentheses) indicate that you may add as many variables as you want.
- Type the first letter of the variable that you want to change, followed by the value you select.
- Press **Enter** again to leave the Alter Menu.

Description of Characteristics

length Number of bits in a data character. The length must match the length expected by the remote system.

Options: 7 or 8.

Default: 8

stop Number of stop bits appended to a character to signal the end of that character during data transmission. This number must match the number of stop bits used by the remote system.

Options: 1 or 2.

Default: 1

parity Checks whether a character was successfully transmitted to or from a remote system. Must match the parity of the remote system.

For example, if you select `even` parity, when the number of 1 bits in the character is odd, the parity bit is turned on to make an even number of 1 bits.

Options: 0 = none, 1 = odd, 2 = even.

Default: 0

rate Bit rate (bits per second). Determines the number of bits transmitted per second. The speed must match the speed of your modem and that of the remote system.

Options: 50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, or 19200.

Default: 1200

device Name of the asynchronous port used to make a connection to a remote system.

Options: tty0 - tty16, or locally created port names. Only the first 8 characters appear in the Alter Menu.

Default: tty0

initial Dial command that must precede the telephone number when you autodial with a modem. Consult your modem user's guide for the proper dial commands.

Options: ATDT, ATDP, etc. Only the first 8 characters appear in the Alter Menu.

Default: ATDT

final Dial command that must follow the telephone number when you autodial with a modem. Consult your modem user's guide for the proper dial command.

Options: 0 = none, valid modem suffix. Only the first 8 characters appear in the Alter Menu.

Default: None

wait Tells the **redial** command in the ATE program to wait *n* seconds between redialing attempts. The wait period does not begin until the connection attempt times out or until you interrupt it. If **attempts** is set to 0, no redial attempt occurs.

Options: 0=none, a positive integer.

Default: 0

attempts Maximum number of times ATE tries to redial to make a connection. If **attempts** is set to 0, no redial attempt occurs.

Options: 0=none, a positive integer.

Default: 0

transfer Type of asynchronous protocol that transfers files during a connection.

Pacing File transfer protocol that controls the data transmission rate by waiting for a specified character or a certain number of seconds between line transmissions. Helps prevent loss of data when the transmission blocks are too large or are sent too quickly for the system to process.

For more information, see "Using Pacing Protocol for File Transfer" on page 8-19.

Xmodem
An 8- bit file transfer protocol that detects data transmission errors and retransmits the data. For more information, see "Using Xmodem Protocol for File Transfer (xmodem)" on page 8-16.

Options: p=pacing, x=xmodem

Default: p

character Specifies the type of pacing to be used.

Character Signal to transmit a line. (Select one character).

Integer Number of seconds the system waits between each line it transmits (select one integer). The default value is 0, indicating a pacing delay of 0 seconds.

Default: 0

Changing (Remapping) the Control Keys

The mapping for these key combinations, set in the default file **ate.def**, follows:

capture key
Starts or stops capturing (saving) the data displayed on your screen during an active connection. This key has a switching or toggle effect.

Options: Any ASCII control character

Default: ASCII octal 002 (STX). This is the **Ctrl-B** key combination on the RT keyboard.

main-menu key
Returns the Connected Main Menu so you can issue a command during an active connection. This control key functions only from the connected state.

Options: Any ASCII control character

Default: ASCII octal 026 (SYN). This is the **Ctrl-V** key combination on the RT keyboard.

previous key
Displays the previous screen any time during the program. The screen that displays varies, depending on the screen in use when you press the previous-screen key.

Options: Any ASCII control character

Default: ASCII octal 022 (DC2). The ASCII control character is mapped to the AIX interrupt signal. This is the **Ctrl-R** key combination on the RT keyboard.

Changing or remapping the control keys may be necessary if you have two applications in which control keys conflict. For example, if the control keys mapped for the ATE program conflict with those in your text editor, you may want to remap the control keys for ATE.

A system user with some programming experience can remap the control keys by editing the default file **ate.def**. See "Changing Your Default File" on page 8-13.

ASCII Control Characters

The ASCII control character you select may be in octal, decimal, or hexadecimal format.

octal	000 through 037. The leading zero is required.
decimal	0 through 31.
hexadecimal	0x00 through 0x1F. The leading 0x is required. The x may be uppercase or lowercase.

Changing Your Default File

The first time you run the ATE program, it creates a file named **ate.def** in the current directory. This file contains default values for:

Data transmission characteristics

Local system features

Dialing directory file

Control keys

The default file that is created the first time ATE runs looks like this:

```
LENGTH          8
STOP            1
PARITY          0
RATE            1200
DEVICE          tty0
INITIAL         ATDT
FINAL
WAIT            0
ATTEMPTS        0
TRANSFER        p
CHARACTER       0
NAME            kapture
LINEFEEDS       0
ECHO            0
VT100           0
WRITE           0
XON/XOFF        0
DIRECTORY       /usr/lib/dir
CAPTURE_KEY     002
MAINMENU_KEY    026
PREVIOUS_KEY    022
```

A5ACG016

A system user with some data processing experience can edit the **ate.def** file with any AIX text editor to change the values of these characteristics.

How to Edit the Default File

1. Leave ATE.
2. Enter the text editor and display the **ate.def** file.
3. Delete all variables which you *do not* want to change.

   This simplifies reading the file, since the system ignores variables that remain the same as the system's defaults.
4. Change the values for those variables you want to change.
5. Save the file.

We now give an example. Assume that you want to change:

RATE to 300 bps.
DEVICE to `tty3`
DIRECTORY to `my.dir`
TRANSFER to `x` (xmodem)

The **ate.def** file that you create looks like the following:

```
RATE          300
DEVICE        tty3
TRANSFER      x
DIRECTORY     my.dir
```

A5ACG011

From now on when you start using ATE, the program looks for a file in the current directory named **ate.def**. Then it reads the values in the file and changes the system's defaults to those you set.

Additional Information

- Variable names must be in uppercase and spelled exactly as they appear in the original default file.
- Type only one variable per line.
- If you enter an incorrect value, you receive a system message but the program continues, using the default value.

Using Xmodem Protocol for File Transfer (xmodem)

Xmodem protocol is an 8-bit transfer protocol that can detect data transmission errors and then retransmit the data. The work station that sends data must wait until the remote system sends a signal that it is ready to receive data.

When the receiver gets data, it returns an acknowledgement to the sender. The ATE receiver times out if it does not receive data 90 seconds after file transfer is initiated.

Xmodem Shell Command

A shell command **xmodem**, shipped with ATE, starts an xmodem file transfer protocol on a remote RT. This command is used in combination with the **send** or **receive** command on the Connected Main Menu.

Sending and receiving with **xmodem** are complementary operations. One system must be set to send while the other is set to receive.

You can interrupt the **xmodem** shell command by pressing the **Ctrl-X** key combination.

Format

`xmodem` *-opt filename*

xmodem	Transfers files between computer systems that have ATE installed.	
-opt	The option you use with the **xmodem** command:	
	-s	Send data to the local RT.
	-r	Receive data from the local RT.
filename	The name of the file you send or receive.	

Sending a File

After a connection to a remote RT is established, you can send a file from the local system to the remote RT using the following procedure.

Sending a File with Xmodem Protocol

1. Type `xmodem -r` *filename.*

   For example, to send `mayfile` to a remote RT type:

   `xmodem -r mayfile`

2. Press **Ctrl-V**.

   The Connected Main Menu displays.

3. Select the **send** command.

   `-s mayfile`

After the file is transferred, the connection screen displays.

Receiving a File

After a connection to a remote RT is established, you can receive a file from the remote RT with the following procedure:

Receiving a File with Xmodem Protocol

1. Type `xmodem -s` *filename*.

   For example, to receive the `infile` from the remote RT, type:

   `xmodem -s infile`

2. Press **Ctrl-V**.

   The Connected Main Menu displays.

3. Select the **receive** command.

   `-r infile`

After the file is transferred, the connection screen displays.

Using Pacing Protocol for File Transfer

Pacing is a file transfer protocol required by some systems. It controls data transmission by waiting for a specified character or waiting a specified number of seconds between lines. This protocol prevents the loss of data when the block size is too large or data is sent too quickly for the system to process.

Use this protocol to send text files only. Files containing embedded control characters cannot be transferred.

Asynchronous Terminal Emulation supports two types of pacing control protocols: character pacing and interval pacing.

Character Pacing

The **send** protocol transmits data from the file until it finds a linefeed. It then sends out a carriage return and waits to receive the pacing character before sending more data. It times out if it does not receive the correct pacing character in 30 seconds.

The **receive** routine sends a pacing character as soon as it is started. It times out if it does not receive data within 30 seconds.

When data transmission starts, the **receive** routine sends a pacing character to the sender whenever it finds a carriage return in the data. The program ends when it receives nothing for 30 seconds.

Interval Pacing

The **send** routine begins to transfer data as soon as it is started. When it finds a linefeed, it converts it to a carriage return and waits a specified time before it sends the next line.

The **receive** routine looks for data as soon as it is started, and times out if it does not receive data within 30 seconds.

Additional Information

UNIX ®files traditionally contain only linefeeds (called ***newlines***) to delimit the end of a line. DOS or CP/M files usually use only carriage returns. For this reason, the pacing **send** protocol converts linefeeds to carriage returns. The pacing **receive** protocol converts

---

® UNIX is a registered trademark of AT&T.

any linefeed/carriage return combinations or single carriage returns to linefeeds before saving the file.

Chapter 9. Managing the Basic Networking Utilities

CONTENTS

About This Chapter

The Basic Networking Utilities (BNU) program is part of the Extended Services facility of the AIX operating system. Composed of directories, files, user commands, and administrative programs and daemons, BNU enables users to communicate with computers other than their local RTs. This chapter provides the information you need to manage the BNU software.

The chapter begins with a brief overview of the hardware required to use the BNU facility, and then continues with a more extensive description of the BNU software. It lists and briefly describes the directories, data base files, administrative files, user commands, administrator commands, and administrative daemons that comprise the networking utilities.

The system administrator responsible for the BNU facility must perform the following initial administrative tasks:

- Installing the software
- Setting up the remote communications facilities.

Note: If your site uses the Transmission Control Protocol/Internet Protocol (TCP/IP), you will need to perform some additional steps in order for BNU and TCP/IP to communicate.

This chapter also explains the routine tasks involved in maintaining the software. These include the following:

- Monitoring file transfers
- Cleaning up the spooling directories
- Working with log files.

The BNU software includes four daemons that handle remote communications activities such as the following:

- Transporting copy requests
- Scheduling work in the spooling directories
- Executing remote commands
- Communicating with TCP/IP.

This chapter also includes information about running automatic maintenance routines.

The final sections of this chapter describe how to handle some common problems encountered when using the BNU software.

Overview of the BNU Hardware

Before a local computer can communicate with a remote computer, a two-way communication link must be established between the systems. There are two ways to set up such a link:

- Using a hardwired line with a device such as tty
- Using a telephone line with a modem
- Using a TCP/IP connection over a Token Ring or Ethernet network.

The hardwired connection links a port on the local computer to a port on the remote computer. A hardwired line is advantageous when users on local systems communicate frequently with remote systems; the link is always available and access time is short. However, a port used for a hardwired communications link is not available for any other purpose.

A hardwired connection is made over an RS-232C or RS-422 serial port at transmission rates of up to 19,200 bits per second. The recommended length of such direct links is 50 feet or less because noise becomes a problem with greater distances. It is possible to obtain longer lengths by using a lower transmission rate and/or limited distance modems (short-haul modems) at both ends of the link.

The second type of connection uses a telephone line and a modem. In this case, the user on the local computer establishes the connection to a remote computer through an Automatic Calling Unit (ACU), also referred to as an autodialer or a modem. The modem attached to the remote system answers the telephone, and the communications software then completes the connection.

The advantage of a modem connection using a phone line is that the local and remote ports are not dedicated to a single computer. The disadvantage is that the port of the remote system may be busy. A dial-up link also requires additional software and hardware, such as the ACU, that is not necessary with a hardwired connection.

The third type of connection uses TCP/IP. This connection requires an optional adapter for the RT. Depending on the type of network used at your site, either a Token Ring adapter or a Baseband (Ethernet) adapter is required. "Setting Up BNU for Use with TCP/IP" on page 9-67 contains information about using TCP/IP with the BNU facility.

For additional information about the hardware used in BNU communications, see "Managing Ports, Cables, and Modems" on page 5-39.

Overview of the BNU Software

The Basic Networking Utilities software is composed of the following items:

- Directories in which the various files and programs are stored
- Files in the supporting data base containing information needed to establish remote connections and determine access permissions
- Administrative files used primarily in transferring data between computers
- User commands that perform the basic BNU functions
- Administrator commands comprised of programs that enable you to configure and maintain BNU
- Daemon programs that handle file transfers, communications with TCP/IP, scheduling work, and remote command executions.

Basic Directories

There are several basic directories that contain the BNU programs and support files. Some of these directories are unique to BNU, while others are common to the AIX operating system. The descriptions that follow are intended only as brief overviews. The remaining sections of this chapter discuss the contents of these directories in more detail.

The /etc/locks Directory

This directory contains the lock (LCK.*) files for the devices used by BNU in establishing remote connections. Lock files prevent several users from accessing a specified device at the same time.

The /usr/adm/uucp Directory

This directory contains the files comprising the BNU supporting data base, several versions of the command **uutry**, and four scripts for automatic routine maintenance operations.

The files in the supporting data base include the following:

Devices	**Permissions**
Dialcodes	**Poll**
Dialers	**remote.unknown**
Maxuuscheds	**Systems**
Maxuuxqts	Cvt

The routine maintenance scripts include the following:

uudemon.admin **uudemon.hour**
uudemon.cleanu **uudemon.poll**

The /usr/bin Directory

This directory includes many of the AIX operating system executable programs. It also contains the BNU executable files for the following commands:

ct **uupick**
cu **uustat**
uucp **uuto**
uulog **uuname**
uux

The /usr/lib/uucp Directory

This directory contains two administrator commands:

uucheck
uucleanup

It also includes the following BNU ***daemons***, programs that handle file transfers, communications with TCP/IP, scheduling work requests, and remote command executions:

uucico **uusched**
uucpd **uuxqt**

The /usr/spool/cron/crontabs Directory

This is a standard AIX directory that contains all users' **crontabs (cron** tables) files. The **cron** daemon automatically runs commands at specified dates and times according to the instructions included in these **crontabs** files.

One of the files in this directory is named **uucp**. It contains instructions to run designated BNU commands at specified times in order to perform certain administrative tasks automatically. These tasks include processes such as handling file transfers and cleaning up the files in the spooling directories.

The /usr/spool/uucp/*system_name* Directory

This is a spooling directory on the local system. It contains queued requests issued by local users for file transfers to remote systems and for command executions on remote systems.

The /usr/spool/uucppublic Directory

This is the public directory for the BNU facility, and one of these directories exists on every system connected by the networking utilities. When a user transfers a file to another system, or issues a request to execute a command on another system, the files generated by these BNU commands are stored in the public directory on the designated

system. However, the user can specify a destination other than the public directory when issuing the **uucp**, **uuto**, or **uux** commands.

For security reasons, you may prefer to set up restrictions that limit the destination of transferred files to the public directory.

Files in the Supporting Data Base

The files in the supporting data base contain information that BNU requires to perform the following tasks:

- Establish connections to remote computers
- Determine and modify access permissions.

In terms of establishing links to remote computers, a process discussed in "Setting Up Remote Communications" on page 9-20, the most important of these files are the **Devices**, **Permissions**, and **Systems** files.

The Devices File

The file **/usr/adm/uucp/Devices** contains information about the devices on the local system that can establish a connection to a remote computer. This file includes information for hardwired, telephone, and TCP/IP communication links.

For detailed information about customizing the **Devices** file for your site, see "Setting Up Devices" on page 9-22.

The Dialcodes File

The file **/usr/adm/uucp/Dialcodes** contains the initial digits of telephone numbers used to establish communications over a phone line. These numbers, such as the code for an outside line, the area code, and the three-digit exchange, are called ***dialing code abbreviations***. The **Systems** file, discussed below, uses these dialing code abbreviations as part of the telephone entry when attempting to contact the modem on a remote computer.

For information about customizing the **Dialcodes** file for your site, see "Setting Up Dialing Codes" on page 9-42.

The Dialers File

The file **/usr/adm/uucp/Dialers** includes a line of characters containing information about the way in which every modem connected to the local system acts when contacting a remote system. Each line in this file specifies the "initial handshaking" that determines whether the communications link between the two systems is set up correctly and ready to transfer data.

For information about customizing the **Dialers** file for use at your site, see "Setting Up Dialers" on page 9-30.

The Maxuuscheds File

The file **/usr/adm/uucp/Maxuuscheds** limits the number of remote systems that the **uucico** program (which handles file transfers) can contact at any one time. This file is used in conjunction with the **uusched** daemon and the lock files in the **/etc/locks** directory to determine the number of systems currently being polled.

You do not need to configure the **Maxuuscheds** file. You simply use it to help manage your system resources and load averages. This file generally requires little or no maintenance unless the system on which it is installed is utilized frequently and heavily by users on remote systems.

For additional information about the **Maxuuscheds** file, see "Limiting the Number of Scheduled Jobs" on page 9-96.

The Maxuuxqts File

The file **/usr/adm/uucp/Maxuuxqts** limits the number of **uuxqt** processes running simultaneously on a local system. (The **uuxqt** daemon executes commands on a local system that have been issued by users on a remote system.)

You do not need to configure the **Maxuuxqts** file, which generally requires little or no maintenance unless the system on which it is installed is utilized frequently and heavily by users on remote systems.

For additional information about the **Maxuuxqts** file, see "Limiting the Number of Remote Executions" on page 9-92.

The Permissions File

The file **/usr/adm/uucp/Permissions** contains information that specifies the remote systems that are allowed to communicate with the local system. This file has options that enable the system manager to specify the remote computers that may log in to the local system, whether these systems may send and receive files, and the commands that users on the remote computer may run on the local system.

For detailed information about setting up the **Permissions** file to meet the special needs of your group of users, see "Customizing the Permissions File" on page 9-43.

The Poll File

The file **/usr/adm/uucp/Poll** contains information specifying when BNU should "poll" designated remote systems. This file is used in conjunction with the **/usr/spool/cron/crontabs/uucp** file, the **uudemon.hour** script, and the **uudemon.poll** script. Together, these files are responsible for initiating automatic calls to remote systems to perform certain maintenance tasks.

For information about using the **Poll** file, see "Customizing the Poll File" on page 9-58 and "Polling Remote Systems (uudemon.poll)" on page 9-100.

The remote.unknown File

The file **/usr/adm/uucp/remote.unknown** handles requests for connections from remote computers that are not listed in the **Systems** file.

For information about using the **remote.unknown** file, see "Customizing the remote.unknown File" on page 9-58.

The Systems File

The file **/usr/adm/uucp/Systems** contains a list of the remote computers with which users on the local system can communicate. Each entry in the file represents a remote system, and users on the local system cannot communicate with a remote system unless that system is listed in the local **Systems** file.

For information about setting up the **Systems** file for use at your site, see "Customizing the Systems File" on page 9-33.

Administrative Files Used to Transport Data

The primary purpose of the Basic Networking Utilities facility is to send and receive data. BNU uses six types of files to accomplish this task:

command (or work) files	lock files
data files	machine log files
execute files	temporary data files

Command/work (C.*) Files

When a user requests a file transfer or a remote command execution, the **uucp** and **uux** commands create **C.*** files in the spooling directory, **/usr/spool/uucp**/*system_name*, on the local system. (The *system_name* entry in the path name represents the computer to or from which the files are transferred, or the computer where the command is to be executed.)

Command files are often referred to as "work" files because they contain information necessary to perform the requested work. Essentially, they provide the **uucico** daemon (for **uucp**) and the **uuxqt** daemon (for **uux**) with the data required to transmit the source file or to execute the AIX command on the specified computer.

Command files contain the following types of information:

- An S (send) or R (receive) notation
- The full path name of the source file
- The full path name of the destination file
- The sender's login name
- A list of the command options

- The name of the associated data file
- The source file's permissions code ("mode bits")
- The name of the recipient on the remote system.

For more information about these files, see "Additional Information about Command Files" on page 9-86.

Data (D.*) Files

When a user issues the **uucp** command with the **-C** flag to send a file to the spooling directory for transfer, **uucp** creates a data file to contain the actual source file. The **uux** command also creates a data file, which either contains the data for a remote command execution, or becomes an execute file on a remote system for a remote command execution.

For more information about these files, see "Additional Information about Data Files" on page 9-89.

Execute (X.*) Files

Executable files created by **uux** contain the command string that the user wants to run on the remote system. These files contain the following types of information:

- the user line, which includes the user's login and the name of the local system
- an error status return line
- the name of the user requesting the command execution
- the name(s) of any file(s) required to execute the command
- the name of the system and file designated to receive standard output from the command execution
- the command string itself.

For more information about these files, see "Additional Information about Execute Files" on page 9-92.

Lock (LCK.*) Files

Every time a user attempts to contact a remote system using a device specified in the **Devices** file, BNU creates a lock file in the directory **/etc/locks**. This temporary file "locks" the device so that another user cannot access it while it is in use.

For more information about these files, see "Additional Information about Lock Files" on page 9-30.

Machine Log Files

BNU creates a log file on the local system, in the appropriate machine-specific subdirectory under **/usr/spool/uucp/.Log**, for each remote system with which it communicates. At specified intervals, the **uucleanup** command combines these log files

and stores them in a directory named **/usr/spool/uucp/.Old**. The combined files remain in this directory for a specified time period, after which they are removed.

For more information about these files, see "Working with Log Files" on page 9-79.

Temporary (TM.*) Data Files

BNU transfers a data file from the spooling directory on a local system to the public spooling directory on a remote system, placing the original data file in a temporary data file for further action. If there are no transfer problems, BNU renames the temporary file with the appropriate **D.*** name and sends it on to the specified destination. If the processing terminates abnormally, the temporary file remains in the spooling directory until you remove it either manually, or with an automatic cleanup procedure.

For more information about these files, see "Additional Information about Data Files" on page 9-89.

User Commands

BNU has a number of commands intended primarily for users, although system administrators will find that several of these commands are useful in managing the networking facility.

For detailed information about these commands, which are stored in the **/usr/bin** directory, refer to the BNU chapter in *Using the AIX Operating System*, and to the descriptions of the individual commands in *AIX Operating System Commands Reference*.

The ct Command

This program instructs the local computer to use the telephone network to call a modem attached to a remote terminal. When the remote modem answers, the **ct** command issues a login process to the remote terminal, allowing the user on the remote terminal to log in to and perform tasks on the local system.

The cu Command

This program connects a local system to a remote system. The user can then execute commands on either computer without dropping the communications link. If the remote system is also running under a UNIX-based operating system such as AIX, a user can transfer files between the two systems.

The uucp Command

This program enables a user to copy files from one system to another system. The **uucp** command creates command and data files as necessary, and then calls the **uucico** daemon to deliver the files.

For additional information about managing **uucp**, see "Transporting Copy Requests" on page 9-82.

The uulog Command

This command displays the contents of a log file of **uucico** or **uuxqt** activities for a specified system. BNU creates individual log files for each remote system with which a local system communicates using the **uucp**, **uuto**, and **uux** programs.

For information about using **uulog**, see "Working with Log Files" on page 9-79.

The uuname Command

This program identifies compatible remote systems with which a user can communicate using BNU. The **uuname** command displays a list of all the computers networked to the local system.

For additional information about using **uuname** in system management, see "Checking Networked Systems (uuname)" on page 9-67

The uupick Command

This program retrieves files transferred with the **uuto** command, which places the files in **/usr/spool/uucppublic**, the public directory on the local system. The user then issues **uupick** with the appropriate option to receive and handle the transferred files.

The uustat Command

This program displays information about the status of transfers requested by the **uucp** and **uuto** commands, and about the status of remote command executions requested by **uux**. The **uustat** command also gives the user limited control over BNU jobs queued to run on remote systems. Using **uustat** with the appropriate flags, a user can perform the following actions:

- Check the general status of connections to other systems
- Cancel BNU job requests.

For additional information about using **uustat** in system management, see "Performing Routine Maintenance Tasks" on page 9-70.

The uuto Command

This program uses the **uucp** command to transfer files to a specified user on another system. The **uuto** command places the copied file in the public directory (**/usr/spool/uucppublic**) on the recipient's system and notifies the recipient that the file has arrived. The recipient then uses the **uupick** command to receive and handle the file.

The uux Command

This program runs a specified AIX command on a specified AIX system while enabling the user to continue working on the local system. The **uux** command creates command, data, and execute files in the spooling directory on the local system, and then calls the **uucico** daemon to contact the designated system and deliver the files. The **uuxqt** daemon then executes the requested command on the designated system.

For additional information about managing **uux**, see "Executing Remote Commands" on page 9-90.

Administrator Commands

BNU has a number of commands intended specifically for system managers. These are actually programs that help the system manager perform both initial administrative tasks such as installing the network facility, and routine administrative tasks such as cleaning up outdated files. You must be logged in under the **bnuadm** ID or have superuser privileges to use the administrator commands (see "Setting Up the BNU Manager's Login and Password" on page 9-17 for information about setting up the **bnuadm** login ID).

The Cvt Program

This program, stored in **/usr/adm/uucp**, moves existing UNIX-to-UNIX Copy Program (UUCP) data and command files into new BNU directories. After installing the BNU software, you first issue the **uucheck** command to verify the existence of the appropriate BNU file system structure. Then, if you have an existing UUCP facility running on the workstation and you want to save those command and data files, you issue **Cvt** to move the old files into the appropriate new directories.

For information about using **Cvt**, see "Moving Existing UUCP Files into BNU Directories (Cvt)" on page 9-19.

The uucheck Program

This program, stored in **/usr/lib/uucp**, checks for the presence of directories, programs, and files required to operate BNU. The **uucheck** command also checks for certain errors in the **Permissions** file. The system manager should issue **uucheck** after installing BNU, and again after setting up the customized access permissions.

For information about using **uucheck**, see "Checking for Required Directories and Files (uucheck)" on page 9-17 and "Checking for Correct Permissions" on page 9-66.

The uucleanup Program

This program, stored in **/usr/lib/uucp**, has several functions, all associated with removing outdated files from the **/usr/spool/uucp** directory. It is normally executed automatically at specified times by the shell **uudemon.cleanup**, which is started periodically by instructions in the file **/usr/spool/cron/crontabs/uucp**. You can also issue **uucleanup** manually if you have superuser privileges.

For information about using **uucleanup**, see "Cleaning Up the Spooling Directories" on page 9-75.

The Uutry Program

This program, stored in **/usr/adm/uucp**, contacts a specified system with debugging turned on. Users as well as system managers can issue this command, which provides a method of checking call processing capabilities with debugging output. (This output is both displayed on the screen of the local system and written to a specified file.) The **Uutry** command (note the uppercase "U") invokes the **uucico** daemon, which in turn establishes the actual connection to the remote system and transfers the file you are sending.

For information about using **Uutry**, see "The Uutry Command" on page 9-71.

Administrative Daemons

The BNU software includes a number of daemons, programs that the networking facility executes to handle file transfers, communications with TCP/IP, scheduling work requests, and remote command executions. These daemons are generally started automatically, either by a command or by a shell script. However, the system manager (or someone with superuser privileges) can start them manually, if necessary. The four BNU daemons are stored in **/usr/lib/uucp**.

The uucico Daemon

This program, which is the primary BNU daemon, transports the files required to send data from one AIX system to another AIX system. Both the **uucp** and **uux** commands start **uucico** to transfer command, data, and execute files to the designated system. The **uucico** daemon is also started periodically by the BNU scheduler, **uusched**, which handles transferring files queued in the local spooling directory.

For detailed information about using **uucico**, see "Transporting Copy Requests" on page 9-82 and "Executing Remote Commands" on page 9-90.

The uucpd Daemon

This program enables the system manager to establish communications with a remote computer by means of the Transmission Control Protocol/Internet Protocol (TCP/IP). With BNU, users can perform tasks such as transferring files between two systems, or executing AIX commands on remote systems linked over either a hardwired line, or a telephone line attached to a modem. In the same way, they can transfer files and execute remote commands on systems linked via TCP/IP.

- For information about using BNU in conjunction with TCP/IP, see "Setting Up BNU for Use with TCP/IP" on page 9-67.
- For information about using the Interface Program for use with TCP/IP, refer to the TCP/IP chapter in *Using the AIX Operating System.*
- For detailed information about TCP/IP, see *Interface Program for use with TCP/IP.*

The uusched Daemon

This program schedules the transfer of files that are queued in the spooling directory, **/usr/spool/uucp**, on the local system. The **uusched** daemon randomizes jobs in the queue and then starts the **uucico** daemon, which actually transfers the files.

For detailed information about using **uusched**, see "Scheduling Work in the Spooling Directory" on page 9-95.

The uuxqt Daemon

This program executes a command on a designated system. A user issues the **uux** command to run a specified AIX command on a designated system. After creating the necessary files, **uux** starts the **uucico** daemon, which transfers those files to the public spooling directory on the specified system. The **uuxqt** daemon on every networked system periodically searches the spool directory for command-execution requests. When it locates such a request, **uuxqt** checks for necessary files and permissions and then executes the specified command.

For detailed information about using **uuxqt**, see "Executing Remote Commands" on page 9-90.

Performing Initial Administrative Tasks

Before users can take advantage of the Basic Networking Utilities, the system manager must perform certain initial tasks:

- Install the BNU software
- Customize the files used in remote communications.

If your site uses TCP/IP to handle remote communications, you must also configure BNU so that the two programs can work together.

Installing BNU

Installing the networking utilities consists of the following steps:

1. Copying BNU from diskette onto the RT fixed disk (this occurs automatically when you select the BNU option during the installation of the Extended Services facility)
2. Checking for required directories and files
3. Setting up the system administrator login and password
4. Moving existing UNIX-to-UNIX Copy Program (UUCP) files into the new BNU directories.

Warning: If your site has been using a version of the UUCP facility, the following installation will copy the BNU files, directories, and programs over your existing UUCP files, directories, and programs. However, the BNU installation will *not* destroy your existing UUCP command and data files.

If you want to use these existing files with the BNU facility, continue with the installation and then issue the **Cvt** command, discussed in "Moving Existing UUCP Files into BNU Directories (Cvt)" on page 9-19.

Copying the Software onto the Fixed Disk

BNU is part of the Extended Services feature of the AIX Operating System. The first step in installing the networking utilities is to copy the BNU programs from the Extended Services diskette onto the fixed disk of your RT.

1. Log in with the **su** command and enter the superuser password required at your site.
2. To install just the BNU component on your system, select the appropriate option from the Install menu.
3. Now follow the instructions that appear on the screen.

For additional information about installing the Extended Services options, see *Installing and Customizing the AIX Operating System.*

Checking for Required Directories and Files (uucheck)

After installing the BNU software, you should issue the **uucheck** command. This command scans the data copied to the RT from the Extended Services diskette, verifying that the directories, programs, and support files required to operate the networking facility were copied successfully to the fixed disk.

Issue **uucheck** with the **-v** flag, which provides an explanation of the way in which BNU checks the file structure:

```
uucheck -v
```

When the checking process is complete, the shell prompt ($) returns to the screen.

If **uucheck** reports any errors, repeat the installation process. If you continue to have problems, refer to *Installing and Customizing the AIX Operating System.* If you cannot resolve installation problems after consulting *Installing and Customizing the AIX Operating System*, confer with your system support team, or follow the usual procedures at your site for reporting problems.

Note: In addition to issuing **uucheck** at this point, you should also use it after customizing the **Permissions** file because the command examines that file too for certain errors. For detailed information about verifying the **Permissions** entries, see "Checking for Correct Permissions" on page 9-66.

Setting Up the BNU Manager's Login and Password

Anyone with superuser authority can access and edit the BNU programs and files. If you are managing an AIX installation (including BNU) for a relatively small group of users for whom tight security is not an issue, you probably won't find it necessary to set up a separate BNU administrative login. In such a case, the **su** command and its associated password will generally be sufficient for managing the computers at your site. Just remember to protect the superuser password and change it as often as necessary to maintain the required level of security.

However, the installation you're managing may be large enough to require several system administrators, and you may find that time constraints or security requirements necessitate having different individuals maintain different parts of the installation. In that case, you may prefer to limit the number of superusers who can access and modify *all* elements of the software, and create additional administrative logins with the appropriate user ID (UID) and group ID (GID) for individuals responsible for discrete parts of the code.

Note: Administrative logins for users responsible for maintaining the BNU software should normally have a UID of 5 and also a GID of 5. See "Types of User Accounts" on page 2-23 for information about superuser and user accounts.

To set up a BNU administrator's login and password, follow these steps:

1. Enter the **users** command at the superuser prompt (#).
2. Now enter the **add** subcommand to add the new login ID, which is `bnuadm` in the following example:

   `> a u bnuadm`

3. You need to change the user ID, so answer `n` to the `OK? (y)` prompt.
4. Enter the following in response to the `Field?` prompt:

   `Field? uid`

5. Now enter `5` in response to the prompt for the user ID.

   `uid 5`

6. Enter `group` in response to the next `Field?` prompt:

   `Field? gr`

7. Enter `uucp` in response to the `group` prompt:

   `group uucp`

8. You need to enter a password for the new login ID, so answer `n` to the `OK? (y)` prompt.
9. Enter `pa` in response to the `Field?` prompt:

   `Field? pa`

10. Enter the password for the `bnuadm` login ID in response to the `password` prompt.
11. The directory field should contain the following entry:

    `/usr/adm/uucp`

    If it does not, enter `dir` in response to the `Field?` prompt, and then enter the correct name in response to the prompt for a directory.
12. The program field should contain the following entry:

    `/bin/sh`

    If it does not, enter that program name in response to the prompt.
13. To end the procedure, press **Enter** at the next `Field?` prompt. The **users** command displays the new information (the password is encrypted).

14. If the information is correct, press **Enter** after the `OK? (y)` prompt. Press **Enter** again after the prompt `Standard new user initialization? (y)`.

15. Enter the **q** subcommand to exit from the **users** program.

The system adds the new `bnuadm` login ID to the **uucp** entry in the **/etc/group** file, and to the list of users in the **/etc/passwd** file.

- For information about the **/etc/group** file, see "The Group File" on page 2-30.
- For information about the **/etc/passwd** file, see "The /etc/passwd File" on page 2-27.
- For additional information about the **/etc/passwd** file and encrypting passwords, see *AIX Operating System Technical Reference.*

Once you have set up the `bnuadm` login ID and its associated password, go on to the next step in installing BNU.

Moving Existing UUCP Files into BNU Directories (Cvt)

Once the software is successfully installed, you can begin customizing the BNU files for use at your site. Before beginning that process, however, you may need to perform one additional step.

- If your site has *not* been using a version of the UNIX-to-UNIX Copy Program (UUCP) to transfer files, go on to "Setting Up Remote Communications" on page 9-20.
- If your site has been using a version of UUCP, continue reading.

If you are replacing an existing UUCP facility with BNU, you must decide whether you want to include your UUCP data and command files in the new networking file structure. Note, however, that if you do not move your existing UUCP files into BNU, those old files will not run after BNU is installed.

- If you decide you do not need your existing UUCP files, go on to the next section.
- If you decide to incorporate your existing UUCP files into the new BNU directories, you must execute the **Cvt** command, which moves existing UUCP data and command files into the appropriate BNU directories.

The **Cvt** (notice the uppercase "C") shell handles the **C.*** and **D.*** files created under your old UUCP facility so that they will run under BNU. The command first creates any required BNU directories (if they do not already exist) and then transfers the old UUCP files into those directories.

Cvt was designed for use only by the system manager (or someone with superuser authority). Once you issue the command, **Cvt** copies the files automatically; you need do nothing else.

The form of the **Cvt** command follows:

Cvt [*option*] [*filename*]

Cvt has only one option, the **-n** flag, which displays a message about the actions the command performs, but which instructs the command *not* to execute at that time. It's a good idea to issue **Cvt -n** and read the message before you actually move the old UUCP files.

The *filename* entry is optional. If you do not specify one or more file names, BNU transfers all existing UUCP files. However, you may prefer to move only certain files, in which case you can specify them in the command line, using AIX pattern-matching characters as appropriate.

1. When you are ready to move the files, enter the following to transfer all your existing UUCP command and data files into the appropriate BNU directories:

   ```
   Cvt
   ```

2. Now follow the directions on the screen. When BNU displays this prompt,

   ```
   Do you wish to continue (Type y to continue)?
   ```

   enter `y` to transfer the files.

3. If you want to cancel the **Cvt** command, enter `n`.
4. When the transfer is complete, the screen displays the shell prompt.
5. If you receive an error message during this process, execute **Cvt** again. If you continue to receive error messages, follow the usual procedure at your site for reporting software problems.

You are now ready to customize the files in the BNU supporting data base.

Setting Up Remote Communications

In order for BNU to function correctly at your site, you must customize the remote communication facilities by setting up the following:

- A list of the devices used to establish a hardwired communications link or a communications link using a telephone line and a modem
- A list of autodialers (modems) used to contact remote systems via the telephone network
- A list of the remote systems with which the local system can communicate
- An optional list of alphabetic abbreviations representing the prefixes of telephone numbers used to contact the specified remote systems
- The appropriate access permissions specifying the way in which local and remote systems may communicate
- A schedule for monitoring the networked remote systems.

You set up these lists, permissions, schedules, and procedures by modifying the following BNU data base files:

Devices	**Poll**
Dialers	**remote.unknown**
Dialcodes	**Systems**
Permissions	

Note: You will also need to edit the **/usr/spool/cron/crontabs/uucp** file to uncomment the lines that handle the automatic maintenance routines such as cleaning up the spooling directories. See "Running Automatic Maintenance Routines" on page 9-97 for information about these shell scripts. If your site uses TCP/IP, you must also modify the **/etc/rc.tcpip** file. See "Setting Up BNU for Use with TCP/IP" on page 9-67 for information about that task.

In general, BNU establishes a ***hardwired connection*** between a local and a remote system by using these files in the following way:

1. A user on a local system issues a BNU command and the name of a remote system.
2. BNU checks the **/usr/adm/uucp/Systems** file to determine whether the remote system is listed as an entry.
3. If the remote system is listed, BNU checks the **/usr/adm/uucp/Devices** file to determine the specific port, transmission rate, and type of connection ("direct") to use to establish communications with the designated remote system.
4. BNU connects the two systems and, if necessary for the specified command, checks the **Permissions** file on the remote system to determine valid access permissions.
5. If the remote system allows the access, the calling user can log in, transfer files, execute commands, and so on, depending on the BNU command specified in step 1.

BNU generally establishes a ***telephone connection*** (using modems) between two systems by using the data base files in the following way:

1. A user on a local system attached to a modem issues a BNU command with the name of a remote system that is also attached to a modem.
2. BNU checks the **Systems** file for that system.
3. If the remote system is listed and accessible, BNU checks the **Devices** file for specific connection information.
4. From the **Devices** file, BNU gets the type of connection, the port, the transmission rate, the type of modem, and the telephone number.

   Note: BNU gets the phone number associated with the specified system from the **Systems** file. If the system manager has entered a dial-code abbreviation in the **Dialcodes** file, BNU gets the prefix for the number from that listing.
5. All modems listed in the **Devices** file are also listed in a file named **Dialers**. BNU gets the initial handshaking information about the local modem from the **Dialers** file and

contacts the modem on the remote computer using the phone number from the **Systems** file.

6. BNU connects the two systems.
7. If the remote system allows the access, the calling user can log in.

For an example of the way in which the files discussed in the following sections work together to establish a connection between two systems, see "Sample Configuration Files" on page 9-64.

Note: Once you have customized the files discussed in this section, particularly the **Systems** and **Permissions** files, users at your site can send mail messages via the BNU communications facilities. Simply by entering the **mail** command with the appropriate address, a user can send a message to any other user on the local or a remote computer. For additional information about mail, refer to the following:

- For information about the **mail** command, see *AIX Operating System Commands Reference*
- For information about using the mail facility, see *Using the AIX Operating System*
- For information about managing the mail facility, see Chapter 7, "Managing the Electronic Mail System" on page 7-1.

Setting Up Devices

The first, and one of the most important steps in customizing BNU for use at your site is to set up a list of the devices that the networking facility will use to establish connections to remote systems. This task involves editing the file **/usr/adm/uucp/Devices**, which contains data about hardwired, telephone, and TCP/IP communication links.

Note: For detailed information about the hardware used in BNU connections, see "Managing Ports, Cables, and Modems" on page 5-39.

This section includes the following information:

- An explanation of standard entries in a **Devices** file
- A discussion of dialer-token pairs
- Several sample device entries, including connections for both autodialers and hardwired lines (TCP/IP connections are noted in this section and described in more detail in "Setting Up BNU for Use with TCP/IP" on page 9-67.)
- Additional information about lock files.

You must be logged in as the BNU administrator (**bnuadm** is the example login ID), or as **su**, to edit the **Devices** file, which is owned by the **uucp** program login ID.

- For information about setting up a BNU administrator's login, see "Setting Up the BNU Manager's Login and Password" on page 9-17.

- For information about the **uucp** program login ID, see "Setting Up a Systems File" on page 9-39.
- For information about login IDs and their associated passwords, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

Standard Entries in a Devices File

Each entry in a **Devices** file includes the following fields:

- *Caller*, which specifies the type of hardwired or autodialer device
- *Line*, which specifies the device name for the port
- *Line2* depends on the *Caller* entry (seldom used)
- *Class*, which specifies the transmission speed
- *Dialer-Token pairs*, which specify a particular type of autodialer (modem) and the token (a defined string of characters) that is passed to the dialer.

You must have an entry in each of these fields, which are arranged in the following order:

Caller Line Line2 Class Dialer-Token Pairs

A standard entry for a hardwired connection looks like this:

```
Direct  tty0   -   1200
zeus    tty0   -   1200
```

A standard entry for a telephone connection using a modem looks like this:

```
ACU     ltty2   -   1200   hayes
```

The *Caller* Field

You must type one of the following keywords in this field:

Keyword	Explanation
Direct	Use this keyword, which must begin with an uppercase `D`, if your site uses hardwired lines to connect multiple systems.
ACU	Use this keyword, which you must type in uppercase letters, if your site connects multiple systems over the telephone network using Automatic Calling Units (autodialers, or modems).
NETWORK	Type `TCP` (in uppercase letters), if your site uses TCP/IP. (TCP/IP is the network currently used with BNU.)
system_name	Type the name of a particular remote system hardwired to the local system. The *system_name* is the name assigned to each RT, such as `hera`, `merlin`, `opus`, `spock`, and so on.

The *Line* Field

Type the device name for the line, or ***port***, used in the communication link. For example, you can use the appropriate tty device name for a hardwired line, such as `tty1`. For a line connected to an ACU (a modem), again use a device name appropriate to the dialer, such as `tty1` or `tty0`.

The *Line2* Field

If you have typed the ACU keyword in the caller field, and the autodialer in the dialer-token pair is a standard 801 dialer, type the device name of the 801 ACU in this field. For example, if the caller is `ACU` and the line is `tty0`, the *Line2* entry might be `tty1`. If the device type is not 801, you must type a hyphen, or dash (`-`) in this field as a placeholder.

Note: The *Line2* field is used only to support older modems that require 801-type dialers. The modem is plugged into one serial port, and the 801 dialer is plugged into a separate serial port.

The *Class* Field

For a hardwired line, type the transmission rate of the device connecting the two systems.

For a telephone connection, type the speed at which the ACU transmits data, such as `300` or `1200` baud.

Some devices can be used at any speed, in which case you should type the word `Any` (note the uppercase "`A`"). This entry tells BNU to match any speed requested in the **Systems** file.

The *Dialer-Token Pairs* Field

For a hardwired connection, enter the word `direct`.

For a telephone connection, enter the type of dialer and the token that is passed on to that modem. The token is either a telephone number or a predefined string used to reach the dialer.

The Dialer Entry

For a telephone connection, enter one of the following as the dialer:

Entry	Definition
hayes	This is a Hayes dialer.
801	This is a standard 801 autodialer, with a separate 212- or 103-type modem.

OTHER DIALERS — These are other dialers that you can specify by including the relevant information in the **Dialers** file.

NETWORK — This represents a communications network; TCP/IP is the network currently supported by BNU. Type `TCP` here if you have also used `TCP` as the keyword in the caller field.

Each dialer included as part of a dialer-token pair in the **Devices** file is also included as an entry in the **Dialers** file. See "Setting Up Dialers" on page 9-30 for information about the contents of this file.

The Token Entry

The *Token* following the dialer represents either a telephone number or a predefined string used to reach the dialer. If the token represents a telephone number, you can generally leave this part of the field blank, specifying that BNU should use the telephone number listed in the **Systems** file.

Note: Some sites do not include complete telephone numbers in the **Systems** file. Instead, the entry in that file contains only the last four digits of the number, preceded by a dial-code abbreviation. This abbreviation references the remainder of the phone number (such as a 9 for an outside line, the three-digit exchange number, or an access code), which is actually contained in the **Dialcodes** file. However, it is often more efficient to include the complete telephone number in the **Systems** file.

You can also use the characters \D or \T as tokens. The \D string is the default token in a dialer-token pair.

The \D token specifies that BNU should take the phone number listed in the **Systems** file and pass it to the appropriate "dialer script" (entry) in the **Dialers** file *WITHOUT* including the dial-code abbreviation. Leaving the token field blank is the same as entering \D, so a blank is usually sufficient as a token if you have included complete telephone numbers in the **Systems** file.

However, if you are using dial-code abbreviations specified in the **Dialcodes** file for certain telephone numbers, you *MUST* enter the \T string as the token in those entries in the **Dialers** file. The \T token instructs BNU to process the phone number by including the data specified in the **Dialcodes** file.

Sample Entries in a Devices File

The following examples illustrate entries in a **Devices** file for various types of connections. The number preceding the entry is *not* part of the **Devices** file. It is simply a number used to reference that entry in the explanation following the examples. Remember that the first field represents *Caller*, the second *Line*, the third *Line2*, the fourth *Class* , and the fifth the *Dialer-Token pair*. These examples do not include a token; that part of the entry is blank.

```
1. Direct tty0     -   1200
2. zeus   tty0     -   1200
3. Direct tty1     -   1200
4. hera   tty1     -   1200
5. ACU    ltty2    -   1200  hayes
6. ACU    ltty3    -   1200  hayes
7. ACU    ltty3    -    300  hayes
8. ACU    tty4   tty5  1200  801
9. ACU    tty6   tty7   300  801
```

The first four entries specify information about hardwired (`Direct`) connections. The rest of the entries in this sample **Devices** file specify connections made over a telephone line using a modem.

Explanation of Sample Hardwired Connections

Setting up entries for hardwired connections to remote computers is very much like setting up entries for telephone connections. It differs in only two ways:

- It is easier to set up entries for hardwired lines than for telephone connections because the connection itself is simpler. Hardwired connections, by definition, do not require any additional software or hardware, such as a modem.
- In general, each entry for a hardwired connection consists of two parts: the first specifies the port (line) that the BNU command uses to connect to the remote system, which is specified in the second part. However, if the two systems use a permanent virtual circuit connection, the entry comprises a single line in the **Devices** file.

In lines 1 and 2, comprising the first example, the caller field lists `Direct` (for a direct connection) in the first part and `zeus` (the name of the remote system) in the second part. The local system is connected to `zeus` via device `tty0`, which is listed in the line field in both parts of the example:

```
Direct tty0    -    1200
zeus   tty0    -    1200
```

The line2 field contains actual data only when the entry specifies a certain type of telephone connection. A hyphen is used as a placeholder in other types of connections, as

is the case in lines 1 and 2. This tty device transmits at 1200 bps (bits per second), which is listed in the class field in both parts of the example. There is no entry in the dialer-token field because a hardwired connection does not require a modem.

Lines 3 and 4, comprising the second hardwired example, are almost exactly like the entries in lines 1 and 2:

```
Direct tty1    -    1200
hera   tty1    -    1200
```

Again, the caller field lists `Direct` in the first part of the entry and the name of the remote system, `hera`, in the second part. The device name is `tty1` in both line fields, hyphens function as placeholders in the line2 fields, and the tty devices communicates at 1200 bps, as listed in the class field.

Explanation of Sample Autodialer Connections

Notice that in each of these telephone-connection entries, the caller is specified as an autodialer (an ACU, or Automatic Calling Unit). You should use ACU as the caller in all remote connections established over a phone line.

The entry in line 5 is the first autodialer connection in the sample **Devices** file:

```
ACU   1tty2    -    1200  hayes
```

The line field is specified with the device name `tty2`. As was the case with the hardwired entries, a hyphen is used as a placeholder in the line2 field, and the class entry is again a transmission rate of 1200 baud. The dialer part of the dialer-token pair is specified as a `hayes` modem, and the token is left blank.

Note: Remember that a /D string or a blank in the token field tells BNU to use the complete telephone number listed in the **Systems** file. If you are using dial-code abbreviations specified in the **Dialcodes** file, you must enter the /T string as the token. All the autodialer examples in the sample **Devices** file shown above specify complete phone numbers listed in the **Systems** file by leaving the token field blank.

The entries in lines 6 and 7 specify the same modem, a `hayes`, which can be used at either 1200 or 300 baud:

```
ACU    1tty3    -    1200  hayes
ACU    1tty3    -     300  hayes
```

The ACUs are connected to a device named `tty3` (the line field), and the line2 field contains the hyphen placeholder.

In the entries comprising lines 8 and 9, the ACUs are connected to devices named `tty4` and `tty6`, specified in the line fields. In both cases, there is an entry in the line2 field because a standard 801 autodialer is specified in the dialer-token pair:

```
ACU    tty4   tty5  1200  801
ACU    tty6   tty7   300  801
```

Because 801 is specified as the dialer in these two examples, the line2 fields must contain the device names of the 801 ACUs. In this case, both are 212 modems, although `tty6` may be a 103 type. The class entry is a transmission rate of `1200` baud for the first example and `300` for the second. The token part of the dialer-token pair is blank.

Setting Up Entries for Hardwired Connections

The following example demonstrates how to set up a standard **Devices** entry for a hardwired connection:

```
Direct  tty0     -   1200   direct
hera    tty0     -   1200   direct
```

1. Type the keyword `Direct`, with an uppercase D, in the *Caller* field in the first part of the entry.
2. For the caller in the second part, shown as `hera` in the example, type the name of the remote system to which you want to connect the local computer over the hardwired line.
3. In this example, the *Line* (or port) entry is a variation of the standard tty device, `tty0`. Type the device name appropriate for the hardwired connection used at your site in the line fields in both parts of the entry.
4. You must make an entry in each field, so type a hyphen for a placeholder in the *Line2* fields in both parts of the entry.
5. The `tty0` device uses a `1200` bps transmission rate. Type the transmission rate appropriate for the hardwired connection used at your site in the *Class* fields in both parts of the entry.
6. This is a direct connection, so you enter `direct` (all lowercase) in the *Dialer-Token pairs* fields in both parts of the entry.
7. Using the sample entry as a model, continue adding entries to the **Devices** file until you have listed each hardwired device connecting the local system to a remote system.

The following examples show two additional types of entries for hardwired lines:

```
Direct tty8     -   9600   direct
thor   tty8     -   9600   direct
odin   x25.cb   -   9600   direct
```

The first is a two-part entry for a hardwired connection to system `thor`. The first part specifies a `Direct` connection in the caller field, and the second part specifies the remote computer. The line entry in both parts is `tty8`, which communicates at `9600` bps.

The third entry connects system `odin` to the local system using an `X.25.cb` permanent virtual circuit connection.

Setting Up an Entry for an Autodialer Connection

The following example demonstrates how to set up a standard entry in the **Devices** file for a Hayes modem:

```
ACU  ltty1   -  1200  hayes
```

1. You are using a Hayes modem, so type `ACU` in the *Caller* field.
2. The *Line* field contains the name of the device that is attached to the modem. In this example, the line (or port) is a variation of the standard tty device, `tty1`. Type the device name appropriate for your site.
3. Type a hyphen as a placeholder in the *Line2* field because the ACU is a specific modem rather than a standard 801 dialer.
4. The Hayes modem in this example operates at `1200` baud. In the *Class* field, type the baud rate appropriate for your modem and line (this can be 300, 1200, 2400, or higher, depending on the modem).
5. Enter the name of the modem as the dialer in the *Dialer-Token pair* field. The example entry is `hayes`, for a Hayes modem. If you are planning to include complete phone numbers in the **Systems** file, leave the token field blank, like the example. (A blank instructs BNU to use the default \D token.) If you are planning to use dialing-code abbreviations specified in the **Dialcodes** file, enter the token `\T`.
6. Using the sample entry as a model, continue adding entries to the **Devices** file until you have listed each connection between the local system and a remote system that uses a telephone line and a modem.

- For an example of an entry in a **Devices** file specifying a TCP/IP connection, see "Setting Up BNU for Use with TCP/IP" on page 9-67.
- For detailed information about the devices used in BNU communications, see "Managing Ports, Cables, and Modems" on page 5-39

Additional Information about Lock Files

Every time a user attempts to contact a remote system using a device specified in the **Devices** file, BNU creates a lock file in the directory **/etc/locks**. This temporary file "locks" the device so that another user cannot access it while it is already in use. The full path name of a lock file is either

/etc/locks/LCK..*device_name*

or

/etc/locks/LCK..*system_name*

where *device_name* is the name of a device such as `tty0`, and *system_name* is the name of a system such as `hera`.

BNU uses the first form of the lock-file name whenever a connection to a remote system, established over the specified device, is actually in use. The second form of the lock-file name is used only by the **uucico** program to prevent more than one **uucico** connection at a time to the same remote system.

Under normal circumstances, the software automatically removes a lock file when the user establishes a connection to a remote system. However, if a process executing on the specified device or system does not complete its run (for example, if the computer crashes), the lock file will remain in the **/etc/locks** directory until it is either removed manually, or the system is restarted after a shutdown.

Note: The line **rm -f /etc/locks/*** in the file **/etc/rc** instructs the system to clean out all the files in the **locks** directory every time the system is restarted. If you prefer to delete any remaining lock files manually, simply comment out this line.

Setting Up Dialers

The file **/usr/adm/uucp/Dialers** contains an entry for each autodialer (other than an 801-type dialer or a TCP/IP connection) that is included in the **Devices** file. Every modem is listed on a line by itself, and each line includes a series of expect-send sequences that specify the initial "handshaking" that occurs on the communications link before it is ready to send or receive data. In this way, the local and remote systems confirm that they are compatible and configured to transfer data.

The handshaking data are included in a string that tells the **cu**, **ct**, or **uucico** programs the sequence of characters to use to dial out on a particular type of modem. These characters include entries such as `\d` to specify a delay, `\p` for a pause, `\r` for a carriage return, `\c` for a new line, and so on.

You must be logged in as the BNU administrator (**bnuadm** is the example login ID), or as **su**, to edit the **Dialers** file, which is owned by the **uucp** program login ID.

- For information about setting up an administrator's login, see "Setting Up the BNU Manager's Login and Password" on page 9-17.
- For information about the **uucp** program login ID, see "Setting Up a Systems File" on page 9-39.
- For information about login IDs and their associated passwords, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

Standard Entries in a Dialers File

The following example lists several entries in a typical **Dialers** file:

```
hayes   =,-,  ""  \dAT\r\c OK \pATDT\T\r\c CONNECT
penril  =W-P  ""  \d > s\p9\c )-W\p\r\ds\p9\c-) y/c : \E\TP > 9\c OK
ventel  =&-%  ""  \r\p \r\p-\r\p-$ <K\D%%\r>\c ONLINE!
vadic   =K-K  ""  \005\p *-\005\p-* D\p BER?  \E\D\e \r\c LINE
direct
```

The first field in the **Dialers** file matches the fifth field, the dialer-token pair, in the **Devices** file. This is the type of autodialer (modem) used in the connection. As noted above, every dialer listed in the **Devices** must also be listed in the **Dialers** file, with the exception of 801-type dialers and a TCP/IP connection.

Note: Notice that the last entry in the example above consists only of the word `direct`. This entry indicates that hardwired connections do not require any handshaking, or "dialer negotiations."

The second field consists of two sets of two characters, for a total of four entries. These characters comprise a translation string. In the actual phone number of the remote modem, the first character in each string is mapped to the second character in that set.

This entry generally translates the characters = and - into whatever the dialer uses for "wait for dial tone" and "pause". For example, in the second line of the sample file, the = translates into `W` in the phone number, and the - translates into `P` on a Penril dialer.

The handshaking, which is usually an expect-send sequence of ASCII strings, is given in the remainder of the line. This string is generally used to pass telephone numbers to a modem, or to make a connection to another system on the same data switch as the local system. If the match succeeds, the line in the **Dialers** file is interpreted to perform the dialer negotiations.

If the fifth field in the **Devices** file is not a built-in function, this field serves as an index to the **Dialers** file.

Sample Entry in a Dialers File

The following example interprets the first line in the **Dialers** file shown above. This is a standard entry that you can include in your **Dialers** file, but you may need to modify it for use at your site.

```
hayes   =,-,  ""  \dAT\r\c OK \pATDT\T\r\c CONNECT
```

Following is an explanation of how each entry affects the action of the dialer.

Entry	Action
=,-,	Translate the telephone number. Any `=` represents "wait for dial tone" and any `-` represents "pause".
""	Wait for nothing; continue with the rest of the string.
\dAT	Delay, then send `AT` (the Hayes Attention prefix).
\r\c	Send a carriage return (`r`) followed by a new line (`c`).
OK	Wait for `OK` from the remote modem, signaling that the first part of the string has executed.
\pATDT	Pause (`p`), then send `ATDT`. `AT` is again the Hayes Attention prefix, `D` represents a dialing signal, and `T` represents a touch-tone dial tone.
\T	Send the telephone number, which is specified in the **Systems** file.
\r\c	Send a carriage return and a new line following the number.
CONNECT	Wait for `CONNECT` from the remote modem, signaling that the modems are connected at the baud rate specified in the **Devices** file.

If you need to modify this example for use at your site and are unsure about the appropriate entries in the handshaking string, refer to the documentation that accompanied the modems that you are including in the **Dialers** file.

- For an example of an entry in a **Dialers** file, see "Sample Configuration Files" on page 9-64.
- For information about the hardware used in BNU communications, see "Managing Ports, Cables, and Modems" on page 5-39.

Customizing the Systems File

Each entry in the **/usr/adm/uucp/Systems** file represents a remote system with which the local system can communicate. BNU cannot establish a communications link between the local computer and a remote computer unless the remote system is listed correctly in this file. You must set up a **Systems** file on every computer at your site that uses the BNU facility.

The entries in the **Systems** file include the name of the remote system, the time during which users can establish a connection between the local and the remote system, whether the connection uses a hardwired, telephone, or TCP/IP communications link, the speed at which the line transmits data, the phone number used with a modem, and information required to log in on the remote system.

The mail facility uses the entries in the **Systems** file to send messages to system users; mail is sent only to users on those computers listed in this file. The mail facility also uses the access permissions established in the **Permissions** file (see "Customizing the Permissions File" on page 9-43).

Note: If you have installed and configured the **sendmail** program from the Extended Services diskette, you can also send mail to other remote systems that are either listed in the **/etc/hosts** file, or for which the Internet address is known, as long as these host computers are running a **sendmail** or SMTP daemon. However, you cannot use an alias name for the remote system, even if that alias is defined for the domain name daemon, when you are using BNU to send mail or remote-command execution requests over a TCP/IP connection.

You must be logged in as the BNU administrator (**bnuadm** is the example login ID), or as **su**, to edit the **Systems** file, which is owned by the **uucp** program login ID. Users cannot access this file unless they know the password for one of the valid logins.

- For information about setting up an administrator's login, see "Setting Up the BNU Manager's Login and Password" on page 9-17.
- For information about the **uucp** program login ID, see "Setting Up a Systems File" on page 9-39.
- For information about login IDs and their associated passwords, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

Standard Entries in a Systems File

Each entry in a **Systems** file has the following form:

System_name   Time   Caller   Class   Phone   Login

A standard entry for a hardwired connection between a local and a remote system looks like this:

```
hera  Any  hera  1200  -  login:--login: uucp  word:sysuucp
```

A standard entry for a telephone connection using a modem looks like this:

```
merlin  0830-1730  ACU  1200  123-4567  in:--in: uucp1  word: passuucp
```

The System_name Field

This is the name of the remote computer that was assigned when the system was installed at your site. The assigned name is often an indication of the system's use, such as `cad`, `pubs`, or `dev`, but many system managers prefer to assign names such as `spock`, `hal`, or `mrdata`. In general, system names should be a maximum of seven characters in length. In order to be compatible with some older systems, such names should include only lowercase characters (or digits).

You can list a specific computer in the **Systems** file more than once. Each additional entry for a specific system represents an alternate communication path that BNU will use in sequential order to try and establish a connection between the local and the remote system.

The Time Field

This is a string that indicates the days of the week and the times of day during which users on the local system can communicate with the specified remote system. For example, the string `MoTuTh0800–1730` indicates that local users can contact the specified remote system on Mondays, Tuesdays, and Thursdays from 8 a.m. until 5:30 p.m.

As indicated in the example above, the day part of the entry can be a list including any day or days represented by `Mo`, `Tu`, `We`, `Th`, `Fr`, `Sa`, or `Su`. You may also enter `Wk` if users can contact the remote system on any week day, or `Any` if they can use the remote system on any day of the week including Saturday and Sunday.

Enter the time at which users can contact the remote system as a range of times, using the 24-hour clock notation. For example, if users can communicate with the specified remote system only during the morning hours, type a range such as `0800-1200`. If users can contact the remote computer at any time of day or night, simply leave the time range blank.

You can also specify times during which users cannot communicate with the remote system—that is, you can specify a time range that spans `0000`. For example, typing `0800-0600` means that users can contact the specified system at any time *except* between 6 a.m and 8 a.m. This is useful if you need a free line at a certain time of day in order to use the remote system for administrative purposes.

You can include multiple time fields by using a comma (`,`) as a separator. For example, `Wk1800-0600,Sa,Su` means that users can contact the remote system on any week day at any time except between the hours of 6 p.m. and 6 a.m., and at any time on Saturday and Sunday.

You can also include an optional subfield that specifies the minimum time in minutes between an unsuccessful attempt to reach the remote system and the "retry" time when BNU again attempts to communicate with that system. This subfield is separated from the

rest of the string by a semicolon (;). For example, `Wk1800-0600,Sa,Su;2` indicates that if the first attempt to establish communications fails, BNU should continue to attempt to contact the remote system at two-minute intervals.

Note: If you include this subfield, it overrides the default retry time.

The Caller Field

The available callers are `ACU` for a telephone connection using a modem, `system_name` for a hardwired connection, and `TCP` for a connection using TCP/IP.

If you use `TCP`, there is a subfield associated with the caller that specifies a conversation protocol. The default, which you do not have to type, is **g**. To use a different subfield, enter it with a comma and the letter representing one of the other conversation protocols—either **t** or **e**. These protocols are faster and more efficient than the **g** protocol.

Use either the **t** or **e** protocol to communicate with a site running the AIX version of BNU. Use the **e** protocol for a site running a non-AIX versions of BNU. Use the **t** protocol for sites running the Berkeley version of the UNIX-to-UNIX Copy Program (UUCP). See "Setting Up BNU for Use with TCP/IP" on page 9-67 for additional information about these subfields.

The Class Field

This is the speed at which the specified hardwired or telephone line transmits data. It is generally `300`, `1200`, `2400` or higher for a hardwired device, or `300`, `1200`, or `2400` for a telephone connection.

Unless it is necessary to enter a specific transmission rate in this field, you can always use the word `Any`. This instructs BNU to match any speed that is appropriate for the ACU or system connection that you specified in the caller field.

Note: For a telephone connection, the rate you enter in this field should correspond to the rate you entered in the *Class* field in the **Devices** file for this particular ACU. Do not include a transmission rate for a TCP/IP connection. Instead, type a hyphen (-) as a placeholder.

As was the case with the **Devices** file, you must have an entry in every field of a line in the **System** file. If you do not type a transmission rate in this class field, use a hyphen as a placeholder, as indicated above.

The Phone Field

You must make an entry in this field. If you are using a hardwired connection, type a hyphen as a placeholder.

If this entry represents a telephone connection using a modem, you can type the remote modem's phone number, composed of a three-digit alphabetic or numeric exchange prefix and a four-digit number, in one of two ways:

- Type the complete phone number of the modem. If the system is in your local dialing area, type the seven-digit phone number.

  If the system is not in your local dialing area, you should also include any other numbers required to reach the remote modem. These numbers may include a code for an outside line, any appropriate long-distance access codes, and 1 plus the area code. Then type the seven-digit number of the remote modem.

  Note: This type of entry is the most efficient method of including phone numbers if your site uses only a relatively small number of telephone connections. However, if your site includes a large number of remote connections established via a phone line and a modem, you may prefer to enter those numbers in the manner described below.

- Type an optional alphabetic abbreviation that represents the dialing prefix, and then type the four-digit phone number. If you choose this method, you must make certain that you have included the dialing prefix in the **Dialcodes** file.

  For example, if your site communicates regularly via modems to multiple other systems all located at the same remote site, it is more efficient to use a dial-code abbreviation in the **Systems** file than to type the complete phone number of each remote modem.

  Suppose that it is necessary to dial a 9, an access code, an area code, and a phone number in order to reach a remote modem. You could type a prefix representing these numbers, together with the four-digit modem number, for each remote system listed in the **Systems** file.

  Then, in the **Dialcodes** file, enter the prefix and the numbers associated with it. Note that you need to enter this prefix in the **Dialers** file *only once* for all the remote modems listed in the **Systems** file. See "Setting Up a Systems File" on page 9-39 for an example of this usage, and "Setting Up Dialing Codes" on page 9-42 for information about using dial-code abbreviations.

Note: For callers that are actually switches, the phone field is the token the switch requires to get to the particular computer. The token you enter here is used by the caller functions specified in the **Devices** file.

The Login Field

Before allowing the calling local system to establish a connection, the designated remote system must receive certain login information from the local system. You specify this login information in a series of fields and subfields called *expect-send* characters.

Expect-send Characters in Login Fields

You enter the required login information in the following form:

[*expect send*] . . .

The *expect* field contains characters that the local system expects to receive from the remote system. Once the local system receives those characters, it sends another string of characters that comprise the *send* field.

For example, the first expect field generally contains the remote system's login prompt, and the first send field generally contains the login ID of the remote system. The second expect field contains the remote password prompt, and the second send field contains the remote system's password.

The expect field may include subfields entered in the following form:

expect[—*send*—*expect*] . . .

In this case, the first *expect* still represents the string that the local system expects to receive from the remote system. However, if the local system does not receive (or cannot read) that first expect string, it sends its own string (the *send* string within the brackets) to the remote system. The local system then expects to receive another *expect* string from the remote system.

For example, the expect string may contain the following characters:

```
login:--login:
```

The local system expects to receive the string `login:`. If the remote system sends that string and the local system receives it correctly, BNU goes on to the next field in the expect-send sequence. However, if the local system does not receive `login:`, it sends a null character followed by a new line, and then expects to receive a second `login:` string from the remote computer.

Note: If the remote system does not send an expect string to the local system, you must type two double quotation marks (`""`), representing a null string, in the first expect field. For information about using this form of entry, see "Login Failures" on page 9-103. Also, every time the local system sends a field, it automatically transmits a new line following that *send* field. If you do not want to include this automatic new line, make the last two characters in the send string a backslash and the letter c (`\c`).

There are two special strings you can include in the login sequence. The string `EOT` sends an **EOT** (End of Transmission) character, and the string `BREAK` attempts to send a **BREAK** character.

You may include the following *expect-send* strings in login fields in the **Systems** file:

String	Explanation
\N	Null character
\b	Backspace character
\c	If at the end of a field, suppress the new line that normally follows the characters in a send field. Otherwise, ignore this string.
\d	Delay 2 seconds before sending or reading more characters.
\p	Pause for approximately 1/4 to 1/2 second.
\E	Turn on the echo check (useful in the **Dialers** file).
\e	Turn off the echo check (useful in the **Dialers** file).
\K	Send a **BREAK** character. This is the same as entering BREAK.
\n	New-line character
\r	Carriage return
\s	Space character
\t	Tab character
\\	Backslash character
EOT	**EOT** character. When you enter this string, the system sends two EOT new-line characters.
BREAK	**BREAK** character
\ddd	Collapse the octal digits (ddd) into a single character and send that character.

Sample Entry in a Systems File

Following is a line representing a typical entry in a **Systems** file. Remember that the first field represents the name of the remote system, the second is the time during which users can reach the remote system, the third is the caller (hardwired, telephone, or TCP/IP) used for the connection, the fourth is the transmission rate, the fifth is the phone number of the remote modem, and the sixth is the login sequence.

```
hera  Any  ACU  1200  ch6412  login:--login: uucp  word: sysuucp
```

Setting Up a Systems File

You can set up the entries in your site's **Systems** file based on the model shown in the above example.

1. Enter the *System_name* of the remote system.

   In the example, the name of the remote system is `hera`.

   - Type the name of the remote system with which you want the local system to communicate.

2. Enter the *Time* during which the local system can communicate with the remote system.

   The example entry is `Any` (notice the uppercase `A`). This specifies that users on the local system can contact `hera` on any day of the week at any time. If `Any` is not appropriate for this entry at your site, use `Wk` for any weekday, or a specific day or days (`Mo`, `Tu`, `We`, `Th`, `Fr`, `Sa`, or `Su`). If you want to specify a time period during which users can connect to the remote system, enter a time range using the 24-hour clock notation.

   - For example, if local users can communicate with the remote system only between the hours of 8 a.m. and 5 p.m. on weekdays, type the following string in the *Time* field:

     `Wk0800–1700`

3. Enter the type of *Caller* used to establish the remote communication.

   The caller field in the example contains the string `ACU`. This indicates that the connection is established over a telephone using the appropriate line listed in the **Devices** file and the appropriate modem listed in the **Dialers** file.

   - If you are setting up a hardwired connection to a remote computer, type the name of the remote system in this field. This is the same name that you used in the *System_name* field in step 1.
   - If you are setting up a telephone connection, type `ACU`.
   - If you are setting up a TCP/IP connection, type `TCP`.

4. Enter the *Class* that represents the speed of the transmission.

   In the example, the entry in this field is `1200` (bps). This is the rate at which data is transmitted over the communications link specified in the *Caller* field.

   - Type the transmission rate appropriate to your entry in the caller field. Enter a hyphen (-) as a placeholder for a TCP/IP connection.

5. Enter either a hyphen or a telephone number in the *Phone* field.

   The example entry uses `ACU` in the caller field, so the phone field contains the telephone number of the remote modem, `ch6412`. This is actually only part of the phone number; the `ch` prefix instructs BNU to find the remainder of the phone number in the **Dialcodes** file.

   - If you are setting up an entry for a hardwired or TCP/IP connection, use a hyphen as a placeholder in the phone field.
   - If you are setting up an entry for a telephone connection, you can either type the complete telephone number of the remote modem in the phone field, or you can type a dial-code abbreviation defined in your **Dialcodes** file plus the four-digit phone number.

     See "The Dialer Entry" on page 9-24 for additional information about telephone numbers used to establish communications with a remote modem, and "Setting Up Dialing Codes" on page 9-42 for information about dial-code abbreviations.

6. Enter the *expect-send* strings in the *Login* field.

   Note: The sample entry contains four strings in this field. The actual number of strings can vary, depending on the remote system and the complexity of the login sequence. Four to six strings are generally sufficient for BNU communications between two AIX systems, but, again, that number can vary.

   a. Type the first string that the local system should expect to receive from the remote system. This is generally the remote system's login prompt.

      The first expect string in the sample entry includes the characters `login:--login:`. This entry indicates that the local system expects to receive a string containing the word `login:`, which is the remote system's login prompt. If the local system receives that string, BNU goes on to the next field, the first send characters. If the local system does not receive (or correctly read) the first expect string, it sends a carriage return to the remote system, expecting then to receive a second `login:` string from that system.

      You do not have to enter the complete prompt; you can use part of the string. For example, the sample entry could use the string `in:` rather than the complete string `login:`. However, the partial string must end with the last character that the local system expects to receive from the remote system.

      - Type the login prompt used by the remote system to which you are setting up a connection in the first expect field.

        Note: It is always a good idea to set up this field with the login prompt included a second time in the *send—expect* subfield, as shown in the example.

b. Type the first string that the local system will send to the remote system. This is generally the remote system's login ID.

   The sample entry uses the string `uucp`, which represents the **uucp** program login ID. In general, BNU uses a version of this login ID as the first send string in the *Login* field in the **Systems** file. (However, on some remote computers this send string could contain a login ID that is not a version of **uucp**, depending on the administrative customs at the remote site.) See "Setting Up BNU Login IDs and Passwords" on page 9-59 for more information about login IDs and their associated passwords.

   Note: Users do not need to know about the **uucp** program login ID. When employing the BNU facility to transfer files or execute remote commands, users need only enter the appropriate command with the required system name. The **uucp** program login ID, which appears as an entry only in the **Systems** and **Permissions** files, is used internally by BNU for transporting files.

   - Type the login ID of the remote system, which your local system will transmit in response to the remote system's login prompt.

   For detailed information about setting up login IDs and their associated passwords, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

c. Type the second string that the local system should expect to receive from the remote system. This is generally the remote system's password prompt.

   The example entry uses the string `word:`. The actual remote system prompt used in the example is `password:`, but as indicated in the discussion about the remote system's login prompt, you don't have to type the complete expect string.

   In this case, you could use the complete string `password:`, but you can also use a partial string such as `ssword:` or even `ord:`, as long as the last character in the string is the last character that the local system expects to receive from the remote system. Notice that in the example, the last character in the string is a colon (:).

   - Type the complete or partial string that represents the password prompt of the remote system for which you are setting up the **Systems** entry. Be sure to include, as the last part of that string, any character that follows the password prompt, such as `:`, `>`, `#`, and so on.

d. Type the second string that the local system will send to the remote system. This is generally the remote system's password.

   The sample entry uses the password `sysuucp`. In this example, that is the password associated with the **uucp** program login ID on the remote system.

   Note: The encrypted password associated with the **uucp** program login ID is stored in the **/etc/passwd** file. If security is a consideration at your site, you can change the password for the **uucp** login ID frequently. For information about the **uucp** program login ID, see "Setting Up a Systems File" on page 9-39.

For additional information about login IDs and passwords, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

- Enter the remote system's password, which your local system will send in response to the password prompt.

7. Using the sample entry as a model, continue adding entries to the **Systems** file until you have specified every remote system with which you want the local system to communicate.

Note: Remember that you must set up this type of entry in the **Systems** file on every computer at your site that uses the BNU facility.

- For additional examples of entries in **Systems** files, see "Sample Configuration Files" on page 9-64.
- For examples of using the **uucp** program login ID and variations of that login with TCP/IP, see "Setting Up BNU for Use with TCP/IP" on page 9-67.

Setting Up Dialing Codes

The **Dialcodes** file contains dial-code abbreviations and partial phone numbers that complete the telephone entries in the **Systems** file. If your site uses a large number of remote connections established over phone lines and modems, you may find it useful to set up such dial-code abbreviations.

You must be logged in as the BNU administrator (**bnuadm** is the example login ID), or as **su**, to edit the **Dialcodes** file, which is owned by the **uucp** program login ID.

- For information about setting up an administrator's login, see "Setting Up the BNU Manager's Login and Password" on page 9-17.
- For information about the **uucp** program ID, see "Setting Up a Systems File" on page 9-39.
- For detailed information about setting up other BNU login IDs and passwords, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

Entries in the **Dialcodes** file contain an alphabetic prefix attached to a partial phone number that may include, for example, codes for outside lines, long-distance access codes, area codes, and three-digit exchange numbers. You enter the relevant alphabetic prefix (representing the partial phone number), together with the remaining four digits of that number, in the *Phone* field in the **Systems** file.

If users at your site communicate regularly via telephone lines and modems to multiple systems all located at the same remote site, or to multiple systems located at different remote sites, you may find it more efficient to use dial-code abbreviations in the **Systems** file than to enter the complete phone number of each remote modem in that file.

Note: If your site uses only a relatively small number of telephone connections to remote systems, you will probably find it more efficient to include the complete phone numbers of the remote modems in the **Systems** file than to use dial-code abbreviations.

Suppose that it is necessary to dial a 9, an access code, an area code, and a phone number in order to reach remote modems at a site with which your users communicate on a regular basis. Rather than typing 15 or more digits for each modem at the remote site in the **Systems** file, you can simply enter an alphabetic prefix (set up in the **Dialcodes** file) and the remaining four digits of the phone number for each remote modem.

Following is the form of an entry in a **Dialcodes** file:

abv dialing_sequence

The *abv* part of the entry is an alphabetic prefix, containing up to eight letters, that you establish when you set up the dialing-code listing. The *dialing_sequence* is composed of all the digits in the number that precede the actual four-digit phone number.

Following is an example entry in the **Systems** file that includes a dial-code abbreviation:

```
zork  Any  ACU  1200  boston4567  login:--login: uucp2  word: leather
```

Following is the relevant entry in the **Dialcodes** file for this dialing prefix:

```
boston 9=1617123
```

Note: You need to enter this prefix *only once* in the **Dialcodes** file. Once you have set up a dial-code abbreviation, you can use that prefix in all relevant entries in the **Systems** file.

When the user wants to communicate with system `zork`, he or she simply enters the appropriate command and the system name, such as `cu zork`. The modem attached to the local system contacts the modem attached to `zork`, using the number `9-1-617-123-4567` (the hyphens are optional).

For additional examples of entries in a **Dialcodes** file, see "Sample Configuration Files" on page 9-64.

Customizing the Permissions File

Each computer that uses BNU in your installation requires both a **Systems** and a **Permissions** file entry. The **/usr/adm/uucp/Permissions** file contains information about the ways in which the remote computers listed in the **Systems** file are allowed to carry out **uucico** and **uuxqt** transactions with a local system.

The system manager must set up entries in the **Permissions** file that specify a remote system's login ID, whether that remote system is allowed to send files to and receive files from the local system, and which commands the remote system is permitted to execute on the local system.

Note: The ***access permissions*** that you set in a **Permissions** file affect only remote systems; they do *not* pertain to individual users who work on those remote systems. Permissions limiting **uucico** and **uuxqt** activities restrict the access to a local system of all users on a specified remote system, including the manager of that remote system and individuals with superuser privileges (unless you have set up a special login and password

for such users). Conversely, less restrictive permissions allow more generous access to all users on the specified remote system.

Warning: Entries in a **Permissions** file do NOT affect a remote-system user with a valid login on a specified local computer. Such users can connect to, log in on, and use a system regardless of the restrictions you set up in the local **Permissions** file. A user with a valid login is subject only to the permission codes established for that user's user ID (UID) and group ID (GID).

The **Permissions** file contains the following two types of entries:

- The login IDs of remote systems that are allowed to communicate with the local system, and the access permissions for those remote systems (information about sending and receiving files, and executing commands)
- The names of those remote systems with which the local system can communicate, and, again, the permissions that govern those remote systems' access to the local computer. (There must also be an entry for each of these remote computers in the **Systems** file on the local computer.)

The **Permissions** file includes very restrictive access permissions as the default values for sending and receiving files and executing commands. However, the file also provides options that enable you to change these defaults if you want to allow remote systems to have less restricted access to local systems.

Warning: The access permissions you set in this file affect all BNU communications, including those made through the mail facility and through a TCP/IP connection.

You must be logged in as the BNU system administrator, or as **su**, to access and edit the **Permissions** file, which is owned by the **uucp** program login ID.

- For information about setting up a BNU system administrator login ID, see "Setting Up the BNU Manager's Login and Password" on page 9-17.
- For information about the **uucp** program login ID, see "Setting Up a Systems File" on page 9-39.
- For more information about login IDs and passwords, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

Standard Entries in the Permissions File

Each entry in a **Permissions** file is a logical line. Each logical line is composed of the basic entry (a login ID or the name of a remote system) plus optional entries separated either by a space or a tab. Both the basic and the optional entries are composed of "name/value" pairs—that is, the name of the entry or option followed by an equal sign (=) followed by the value of the entry or option, with no spaces allowed within the pair. Comment lines begin with a pound sign (#) and occupy the entire physical line; blank lines are ignored.

The two types of entries in the **Permissions** file are for LOGNAMEs and MACHINEs.

LOGNAME — This type of entry contains the login ID of, and access permissions for, a remote system that is allowed to conduct **uucico** and **uuxqt** transactions with a local system.

MACHINE — This type of entry contains the names of and access permissions for the remote systems with which the local system is allowed to initiate **uucico** and **uuxqt** transactions.

LOGNAME entries are concerned with operations that occur when a remote system contacts a local system. The calling remote computer must be listed in the **Systems** file on the local computer.

MACHINE entries are concerned with operations that occur when a local system contacts a remote system, although the permissions in this entry still apply to the remote system's access to the calling local system.

A remote system listed in a MACHINE entry uses the login ID specified in a LOGNAME entry to communicate with a local system.

A LOGNAME Entry

A LOGNAME entry specifies one or more login IDs of remote systems that are permitted to log in to the local computer in order to conduct **uucico** and **uuxqt** transactions, and the access permissions for those remote systems. The actual login ID can be any name, although the examples in this chapter use a form of the **uucp** program login ID.

Note: Whatever login ID you choose *must* have both a user ID (UID) and a group ID (GID) of 5.

Following is an example of the simplest, and most restrictive type of LOGNAME entry:

```
LOGNAME=uucp
```

In this example, the local system permits access only to a remote system that logs in using the **uucp** program ID. The entry does not contain any optional name/value pairs, which means that the remote system's access to the local system is restricted to the following default permissions:

- The remote system may not ask to receive any queued files containing work that users on the local system have requested to be executed on the calling remote system.
- The local system may not send queued work to the calling remote system when that system has completed its current operations. Instead, the queued work may be sent only when the local system contacts the remote system.
- The remote system may not send files to (write) or transfer files from (read) any location except the BNU public directory (**/usr/spool/uucppublic/***system_name*) on the local system.
- Users on the remote system may execute only the default commands on the local system. (The ***default command set*** includes only the **rmail** command, which users implicitly execute by issuing the **mail** command.)

Note: A name may appear in only one LOGNAME entry. If you have one entry for the **uucp** program login ID, that single entry is sufficient for all remote systems using that login ID. (You list these systems in the MACHINE entry.) You may have additional entries using other forms of the **uucp** program login ID such as **uucpa** or **uucp1**, as discussed below, but you may not include another **uucp** entry.

The example entry above uses the **uucp** program login ID, which is generally sufficient for **uucico** and **uuxqt** transactions between local and remote computers at most sites. However, you may include alternate versions of the **uucp** program login ID if certain computers at your site require different types of permissions when accessing the local system. For information about such login IDs, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

The following LOGNAME entry includes two login IDs used by remote systems (which are specified in a MACHINE entry) to access the local system `zeus`. Notice that both IDs use a form of the **uucp** program login, that they are separated by a colon, and that there are no spaces in the entry:

```
LOGNAME=uucp:uucp1
```

For additional information about login IDs and their associated passwords, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

A MACHINE Entry

The second type of standard entry in a **Permissions** file is the MACHINE entry. This contains the name of the local system, the names of the remote systems with which the local system is allowed to engage in **uucico** and **uuxqt** transactions, and, again, the access permissions for those remote systems. Following is the simplest kind of MACHINE entry:

```
MACHINE=zeus:hera
```

In this example, `zeus`, the local system, is permitted to communicate with `hera`, the remote system. Notice that the two system names are separated by a colon, and that the entry includes no spaces or tab characters. As was the case in the LOGNAME examples, there are no optional name/value pairs in this entry, indicating that the remote system's access to the local system is limited to the following:

- The remote system may not ask to receive any local-system files queued to run on the calling remote system.
- The local system may not access (read) any files except those in the public directory on the local system.
- The remote system may send (write) files only to the local public directory.
- The remote system may execute only those command in the default command set on the local system.

Like a LOGNAME entry, a MACHINE entry may also include a number of different remote systems:

```
MACHINE=zeus:hera:venus:merlin
```

Remember that you must include the name of the local system in the MACHINE entry. You may find it helpful to list that name first so that you do not forget it. Also, unless you explicitly specify additional options as described in "Options Used in the Permissions File," the default is for the most restrictive permissions in both the LOGNAME and the MACHINE entries.

Options Used in the Permissions File

The **Permissions** file includes a number of options that enable you to control access to local systems by remote systems involved in **uucico** and **uuxqt** transactions. The default permissions are restrictive, but you can change these defaults with one or more of the following options:

REQUEST	**COMMANDS**
SENDFILES	**VALIDATE**
READ, WRITE	**CALLBACK**
NOREAD, NOWRITE	**OTHER**

These options enable you to customize your **Permissions** file in such a way that different remote systems are allowed different types of access to the local system when using the BNU file transport and command execution programs.

The REQUEST Option

This option permits a remote system to ask to receive any queued files containing work that users on the local system have requested to be executed on that remote system. The default is not to allow such requests.

When a remote system contacts the local system in order to transfer files or execute commands, that remote system may also request permission to receive any files queued on the local system for transfer to or execution on that remote system. The following option permits such requests:

```
REQUEST=yes
```

The default, which you do not have to enter, is `REQUEST=no`. This specifies that the remote system cannot ask to receive any work queued for it on the local system. In this case, the local system must contact the remote system before files and execute commands queued on the local system can be transmitted to the remote system.

You can use the `REQUEST=yes` option in both LOGNAME and MACHINE entries in the **Permissions** file. Including this option makes it easy for remote-system users to transfer files to and execute commands on a local system. If security is a consideration at your site, you may prefer to restrict this access so that the local system retains controls of file transfers and command executions initiated by remote systems.

Note: Remember, however, that entries in the **Permissions** file do not affect remote-system users with valid logins on a local system.

The SENDFILES Option

This option permits the local system to send queued work to the calling remote system after that remote system has completed its current **uucico** or **uuxqt** operations. The default is not to allow such transfers of queued local work.

When the remote computer has finished transferring files to or executing commands on a local system, that local system may try to send queued work to the calling remote system. This work generally includes files to be transferred to the remote system, or command requests waiting for execution on the remote system. The following option permits this type of operation:

```
SENDFILES=yes
```

This option, which you include only in a LOGNAME entry in a **Permissions** file, allows the transfer of queued work from the local to the remote system once the remote system has completed its operations.

The default value, which you do not have to enter, has the following form:

```
SENDFILES=call
```

This specifies that local files queued to run on the remote system will be sent only when the local system contacts the remote computer. As was the case with the **REQUEST** option, security considerations at your site may require that you limit a remote systems's access to a local system by using the default value for this option. (Remember that you do not have to enter default values.) Remember, too, that entries in the **Permissions** file do not affect remote-system users with valid logins on a local system.

The READ and WRITE Options

These options specify the path names of locations accessible to the **uucico** daemon when transferring files to or from the local system. The default location for both the **READ** and **WRITE** options is the BNU public directory on the local system:

```
READ=/usr/spool/uucppublic WRITE=/usr/spool/uucppublic
```

You can use the slash (/) that represents the **root** directory on the local system as the value part of the name/value pair in a **READ** or **WRITE** option. For example,

```
READ=/ WRITE=/
```

specifies that **uucico** (the daemon that transfers **uucp** and **uux** requests) can read from or write to any file on the local system under the **root** directory that allows access by a user with **other** permissions.

Note: The source or destination file or directory *must* be readable and/or writable to the **other** group. You set these permissions with the **chmod** command. A user who is not logged in with **su** can take away permissions granted by the **READ** and **WRITE** options, but that user cannot grant permissions that are denied by these options.

You can specify more than one path for **uucico** activities, as in the following example:

```
WRITE=/usr/spool/uucppublic:/usr/news
```

This entry permits **uucico** to send files to both the BNU public directory and the **/usr/news** directory.

If you do not specify path names in the **READ** and **WRITE** options, BNU permits files to be transferred only to the **/usr/spool/uucppublic** directory. However, if you decide to specify path names in these options, you *must* enter the path name for every source and destination. For example, if you enter

```
WRITE=/usr/news
```

then the **uucico** daemon would be able to transfer files *only* to that directory. If you enter any path name in either option, you must also explicitly specify the public directory if you want **uucico** to be allowed to place files in that location.

You can include **READ** and **WRITE** options in both LOGNAME and MACHINE entries.

The NOREAD and NOWRITE Options

These options specify exceptions to the **READ** and **WRITE** options.

For example, the following entry,

```
READ=/ NOREAD=/etc WRITE=/usr/spool/uucppublic
```

permits the remote system to read any file on the local system *except* those in the **etc** directory and its subdirectories. The **WRITE** option in this example allows the remote system to transfer files only to the BNU public directory on the local system.

The **NOWRITE** option functions in exactly the same way as the **NOREAD** option—it explicitly specifies directories and files on the local system to which the remote system cannot transfer work.

Note: The specifications you enter with the **READ**, **WRITE**, **NOREAD**, and **NOWRITE** options can help determine the security of your local system in terms of **uucico** transactions.

The COMMANDS Option

This option, which you include only in a MACHINE entry, specifies the commands that remote systems listed in that MACHINE entry can execute on the local system.

Warning: The **COMMANDS** option can jeopardize the security of your system. Use it with extreme care.

The default for this option severely limits the commands that remote systems can execute on the local system:

```
COMMANDS=rmail
```

This means that remote systems can run only the `rmail` command on the local system. (Remember that users enter the **mail** command, which then calls **rmail**.)

When you enter the **COMMANDS** option in the MACHINE part of an entry in the **Permissions** file, the commands you specify in that option override the default. For example, the entry

```
MACHINE=zeus:hera:venus:merlin COMMANDS=rmail:print
```

specifies that the remote systems `hera`, `venus`, and `merlin` can execute both the `rmail` (**mail**) and the `print` commands on `zeus`, the local system. These commands now comprise the ***default command set*** for the remote systems listed in the MACHINE entry.

You can also specify path names to those locations on the local system where commands issued by users on remote systems are stored. For example,

```
COMMANDS=rmail:/bin/print
```

means that in addition to the **rmail** command, remote systems can also execute the `print` command, which is stored in the **/bin** directory. This option is useful when the default path of the **uuxqt** daemon does not include a particular directory where a permitted command resides.

Note: The default path of the **uuxqt** daemon includes only the **/bin** and **/usr/bin** directories.

If you want to allow a certain remote system to execute all the available AIX commands on the local system, enter the **COMMANDS** option with the value `ALL`:

```
COMMANDS=ALL
```

This specifies that the default command set available to the designated remote system (or for a particular login ID used by a remote system) includes all the available AIX commands.

The VALIDATE Option

This option provides a certain amount of security when you find it necessary to include commands in the default command set that could potentially cause damage when executed by a remote system on a local system. Use this option, which you specify only in a LOGNAME entry, in conjunction with a **COMMANDS** option that you specify in a relevant MACHINE entry.

The **VALIDATE** option verifies, to a certain degree, the identity of the calling remote computer. Including this option in a LOGNAME entry means that the calling remote system must have a unique login ID and password for file transfers and command executions.

Note: This option is meaningful only when the login ID and password are protected. Giving a remote system a special login and password that provide unlimited file access and remote command execution ability is equivalent to giving any user on that remote system a normal login and password on the local system, unless the special login and password are well protected.

For example, the following entries,

```
LOGNAME=uucp VALIDATE=hera:venus:merlin

MACHINE=zeus:hera:venus:merlin COMMANDS=ALL
```

specify that if one of the remote systems `hera`, `venus`, or `merlin` attempts to log in to the local system, it must use the login ID `uucp` and the password associated with that login. Once the remote system is logged on, users on that remote system can then execute all AIX commands on the local system.

The **VALIDATE** option links a MACHINE entry, which includes a specified **COMMANDS** option, to a LOGNAME entry associated with a privileged login. BNU requires this validating link because the **uuxqt** daemon, which executes commands on the local system that have been requested by users on a remote system, isn't running while the remote system is logged in and therefore doesn't know which remote system sent the execution request.

Each remote system permitted to log in to a local system has its own spooling directory on that local system; only the BNU file transport and command execution programs are

allowed to write to these directories. For example, when the **uucico** daemon transfers execution files from the remote system `hera` to the local system `zeus`, it places these files in the `/usr/spool/uucp/hera` directory on `zeus`.

Then, when the **uuxqt** daemon attempts to execute the specified commands, it determines the name of the calling remote system (`hera`) from the path name of the remote-system spooling directory (`/usr/spool/uucp/hera`). The daemon then checks for that name in a MACHINE entry in the **Permissions** file. The daemon also checks for the commands specified in the **COMMANDS** option in a MACHINE entry to determine whether the requested command may be executed on the local system.

The CALLBACK Option

This option, which you include only in LOGNAME entries, specifies that no **uucico** transactions will occur until the local system contacts the remote system that is attempting to establish a connection.

Following is the default **CALLBACK** option:

```
CALLBACK=no
```

This specifies that the remote system may contact the local system and begin transferring files without the local system initiating the **uucico** operations.

The following option,

```
CALLBACK=yes
```

specifies that the local system must contact the remote system before that remote system may transfer any files to the local system. The default value, **CALLBACK=no**, is generally sufficient for most sites.

Warning: If two systems both include the **CALLBACK=yes** option in their respective **Permissions** files, they will never be able to communicate with each other.

The OTHER (System) Option

This option, which represents a system name in a MACHINE entry, enables you to set up access permissions for remote systems not explicitly specified in the existing MACHINE entries in a **Permissions** file.

This option is useful in the following circumstances:

- When your installation includes a large number of remote systems that regularly regularly conduct **uucico** and **uuxqt** transactions with a local system
- When it is occasionally necessary to change the default command set specified in the **COMMANDS** option in the MACHINE entry.

Rather than creating separate MACHINE entries for each of these numerous remote systems, you can set up one entry, with `OTHER` listed as the MACHINE, that includes the

appropriate AIX commands specified in a **COMMANDS** option entry. Then, when it becomes necessary to change the default command set, you change the list of commands in only one entry rather than in numerous entries. You may also want to specify different (generally more restrictive) option values for these remote systems.

Following is an example of this type of entry:

```
LOGNAME=uucp1

MACHINE=OTHER COMMANDS=rmail:/bin/print:/usr/bin/nroff
```

This specifies that all remote systems using the **uucp1** login ID that are not included in existing MACHINE entries can execute the `rmail` (**mail**), `print`, and `nroff` commands on the local system.

Note: Notice that this example has very restricted access permissions. With the exception of the limited command set, both the LOGNAME and MACHINE entries use the default options, which restrict remote systems' **uucico** and **uuxqt** transactions with a local system. It's a good idea to restrict access permissions when using the **OTHER** option, although you can include any of the available MACHINE options.

Relationship Between LOGNAME and MACHINE Entries

The following example shows the relationship between the LOGNAME and MACHINE entries in a **Permissions** file:

```
LOGNAME=uucp VALIDATE=hera REQUEST=yes SENDFILE=yes READ=/ WRITE=/

MACHINE=zeus:hera REQUEST=yes COMMANDS=ALL READ=/ WRITE=/
```

The remote computer `hera` may engage in the following **uucico** and **uuxqt** transactions with the local system `zeus`:

- The remote systems may request that files be sent from the local system
- The local system may send files to the remote systems
- The remote systems may execute all available AIX commands on the local system
- The remote systems may read from and write to all directories and files under the **root** directory.

  Note: In this example entry, files owned by the **uucp** program login ID, such as the **Systems** file, are accessible by editing programs like **ed**. This means that a user on `hera` can both examine and modify the **Systems** file on `zeus` (if the AIX permission codes specify that the file is writable).

The example above obviously allows unrestricted access to the local system by the remote system listed in the MACHINE entry. If security is a concern at your site, you should

probably set up this type of unrestricted LOGNAME/MACHINE entry on a local system *only* for a remote computer used by a system administrator or members of the **uucp** group.

Set up another LOGNAME/MACHINE entry in the local **Permissions** file for remote machines used by individuals who do not require unlimited access to that local system. Use another version of the **uucp** login ID in the LOGNAME entry. Then list the remote systems with restricted access in the MACHINE entry, and include only those commands that general users should execute on the local system.

- For information about setting up BNU login IDs, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

Combining MACHINE and LOGNAME Entries

You can combine a LOGNAME and a MACHINE entry into one single entry when both parts include the same options.

For example, consider the following entries:

```
LOGNAME=uucp REQUEST=yes SENDFILE=yes READ=/ WRITE=/

MACHINE=zeus:hera REQUEST=yes COMMANDS=ALL READ=/ WRITE=/
```

As you can see, both the LOGNAME and MACHINE entries include the same values for the REQUEST, READ, and WRITE options. You can therefore merge the two parts, as shown in the following example:

```
LOGNAME=uucp MACHINE=zeus:hera REQUEST=yes SENDFILE=yes COMMANDS=ALL \
READ=/ WRITE=/
```

Note: If the physical line representing an entry is too long to fit on the screen, make the last character in that physical line a backslash (\), which indicates continuation, and then type the remainder of the entry on the next line.

Sample Entries in a Permissions File

This section contains examples of several types of LOGNAME and MACHINE entries. You can use these entries as models for similar entries in your **Permissions** file.

1. The first LOGNAME entry specifies a restricted login for remote systems engaged in **uucico** and **uuxqt** transactions with a local system. This entry is intended for remote computers used by individuals who do not require complete access to the local system.
2. The second LOGNAME entry specifies a much less restricted remote-system login on a local system. Such an entry is intended for a system administrator or members of the **uucp** group.
3. The MACHINE entries specify the remote systems with which a local system may communicate, and the access permissions granted to those remote systems.

- For an explanation of how to set up BNU login IDs and passwords, see "Setting Up BNU Login IDs and Passwords" on page 9-59.
- For examples of complete BNU configuration files, see "Sample Configuration Files" on page 9-64.

Setting Up a Restricted Login on a Local System

The first LOGNAME example illustrates an entry for a very restricted login on a local system. This entry specifies the most restricted access permissions for remote systems attempting to transfer files to and from, and execute commands on, a local system:

```
LOGNAME=uucp1
```

This entry specifies that the remote systems logging in on the local system must use the `uucp1` login and its associated password.

Note: Remember that you must set up this `uucp1` login and its associated password with the **users** command, as described in "Setting Up BNU Login IDs and Passwords" on page 9-59.

This LOGNAME entry includes no options, so BNU uses the restrictive default options that permit the following remote-system access to the local system:

- The remote system may not ask to receive any queued files containing work that users on the local system have requested to be executed on the calling remote system.
- The local system may not send queued work to the calling remote system when that system has completed its current operations. Instead, the queued work may be sent only when the local system contacts the remote system.
- The remote system may send (write) files to and transfer (read) files from only the BNU public directory (**/usr/spool/uucppublic/***system_name*) on the local system.
- Users on the remote system may execute only the default commands on the local system. (The default command set includes only the **rmail** command, which users implicitly execute by issuing the **mail** command.)

This entry alone is sufficient to start **uucico** and **uuxqt** transactions between the local system and the remote systems specified in the relevant MACHINE entry. It permits a specified remote system to transfer files to the public directory on the local system.

Note: For information about setting up the MACHINE entry specifying the remote systems that use the `uucp1` login, see "Setting Up a Local System's Access to Remote Systems" on page 9-56.

Setting Up an Unrestricted Remote Login on a Local System

The sample login in this section specifies a broad range of access permissions granted to remote systems logging in with the **uucp** program login ID.

Note: It is ***NOT*** a good idea to distribute this type of login and and its associated password to the general users at your site. Logins of this kind are usually reserved for a tightly controlled group of computers that use a closely protected **Systems** file.

```
LOGNAME=uucp VALIDATE=hera:venus:merlin REQUEST=yes SENDFILES=yes \
READ=/ WRITE=/
```

This entry provides the following permissions when a remote system logs in using the **uucp** program login ID:

- The calling remote system has a valid, unique login and password on the local system (the **VALIDATE** option).
- The remote system may request that files be transferred to it from the local system (the **REQUEST** option).
- Any requests queued on the local system for work on the remote system will be sent to that remote system during the current session. These queued items are requests by local users for file transfers to and command executions on the remote system (the **SENDFILES** option).
- The remote system may request any files that are readable by a user with **other** permissions, as specified with the AIX command **chmod** (the **READ** option).
- The remote system may transfer files to any file or directory that is writable by a user with **other** permissions. This means that the file or directory is writable by a user with neither **owner** nor **group** permissions (the **WRITE** option).

The commands that the remote system can execute on the local system are specified in a relevant MACHINE entry.

Setting Up a Local System's Access to Remote Systems

The LOGNAME examples illustrate entries specifying the ways in which remote systems listed in the MACHINE part of an entry can conduct **uucico** and **uuxqt** transactions with a local system.

Following is an example of a MACHINE entry that specifies the remote systems with which a local system can communicate. This example is related to the `LOGNAME=uucp1` entry set up for computers with restricted access to the local system:

```
MACHINE=zeus:hera:venus:merlin
```

Local system `zeus` can communicate with the remote systems `hera`, `venus`, and `merlin` Remember that you must include the name of the local system in a MACHINE entry.

Note: Remember too that the access permissions that govern `zeus`'s **uucico** and **uuxqt** transactions with the remote systems listed in this MACHINE entry are included in the LOGNAME/MACHINE entries in the **Permissions** files on `hera`, `venus`, and `merlin`.

Notice that there are no options specified in this entry. BNU therefore uses the restricted list of default options to specify the access permissions granted to the remote systems specified in this list:

- The remote system may not ask to receive any queued files containing work that users on the local system have requested to be executed on the calling remote system.
- The remote system may access (read) only those files that are in the public directory (**/usr/spool/uucppublic**) on the local system.
- The remote system may send (write) files only to the public directory on the local system.
- Users on the remote system may execute only the default commands on the local system. The default command set includes only the **rmail** command (the user enters the **mail** command).

The next sample MACHINE entry accompanies the unrestricted remote-system access provided by the `LOGNAME=uucp` program login ID illustrated in "Setting Up an Unrestricted Remote Login on a Local System" on page 9-56.

```
MACHINE=zeus:hera:venus:merlin REQUEST=yes COMMANDS=ALL READ=/ WRITE=/
```

The options in this entry provide the following access to the remote systems `hera`, `venus`, and `merlin`:

- The calling remote system may request that files be transferred to it from the local system (the **REQUEST** option).
- The calling remote system may execute all AIX commands on the local system (the **COMMANDS** option).
- The calling remote system may request any files that are readable by a user with **other** permissions, as specified with the AIX command **chmod** (the **READ** option).
- The remote system may transfer files to any file or directory that is writable by a user with **other** permissions. This means that the file or directory is writable by a user with neither **owner** nor **group** permissions (the **WRITE** option).

For additional examples of entries in a **Permissions** file, see "Sample Configuration Files" on page 9-64.

Customizing the Poll File

The **Poll** file (**/usr/adm/uucp/Poll**) contains information specifying when BNU should poll designated remote computers. This file is used in conjunction with the **/usr/spool/cron/crontabs/uucp** file, the **uudemon.hour** script, and the **uudemon.poll** script. Together, these files are responsible for initiating automatic calls to remote systems to perform certain maintenance tasks.

Each entry in the **Poll** file contains the name of the remote computer followed by a tab character and a sequence of times to poll. You must specify times as digits between 0 and 23.

Following is a standard entry in the **Poll** file:

```
hera <TAB> 0  4  8  12  16  20
```

This entry specifies that the local system will poll the remote system `hera` every four hours. Depending on how the systems at your site are used, you may need to modify the times specified in the **Poll** file.

You must be logged in as the BNU administrator (**bnuadm** is the example login ID), or as **su**, to edit the **Poll** file, which is owned by the **uucp** program login ID.

Note: You may have trouble modifying the **Poll** file with the INed editor, which expands tab characters.

- For information about setting up an administrator's login, see "Setting Up the BNU Manager's Login and Password" on page 9-17.
- For information about the **uucp** program login ID, see "Setting Up a Systems File" on page 9-39.
- For more information about login IDs and passwords, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

Customizing the remote.unknown File

BNU executes the shell script **/usr/adm/uucp/remote.unknown** when a remote computer that is not listed in the local **Systems** file attempts to communicate with that local system. BNU does not permit the unknown remote system to connect with the local system.

Instead, the **remote.unknown** script appends the following type of entry to the file **/usr/adm/uucp/Admin/Foreign**:

```
FOREIGN=/usr/spool/uucp/.Admin/Foreign
echo "'date':call from the system $1"<<FOREIGN
```

You can modify this file to fit the needs of your site. You must be logged in as the BNU administrator (**bnuadm** is the example login ID), or as **su**, to edit the **remote.unknown** file, which is owned by the **uucp** program login ID.

- For information about setting up an administrator's login, see "Setting Up the BNU Manager's Login and Password" on page 9-17.
- For information about the **uucp** program login ID, see "Setting Up a Systems File" on page 9-39.
- For more information about login IDs and passwords, see "Setting Up BNU Login IDs and Passwords."

Setting Up BNU Login IDs and Passwords

As you know, **uucp** is the login ID for the **uucp** file transport programs; the encrypted password for this login is included in the **/etc/passwd** file. All the example entries in "Customizing the Systems File" on page 9-33 and most of the examples in "Customizing the Permissions File" on page 9-43 use the **uucp** program login ID, and the **Systems** examples use the sample password **passuucp**.

If your site is relatively small and security is not a major concern, you will probably need only one version of the **uucp** program login ID and its associated password. In that case, go on to "Creating BNU Login IDs and Passwords" on page 9-60 and follow the instructions in that section.

However, conditions at your site may require that certain remote computers have different types of access to a specific local system. You can meet this need by creating an alternate version of the **uucp** program login ID with its associated password in the **Systems** file on the local computer.

You then include both the regular **uucp** login ID and the alternate version in LOGNAME entries in the local system's **Permissions** file. These entries give the two remote-system login IDs different access to the local system for **uucico** and **uuxqt** operations.

Note: The practice of creating additional, more restrictive **uucp** login IDs is a good idea if security is a consideration at your site.

You should include one entry in the local **Systems** file intended only for the BNU system administrator (or members of the **uucp** group). You then set up an unrestricted **uucp** LOGNAME entry in the local system's **Permissions** file intended only for that administrator. Together, the entry in the **Systems** file and the entry in the **Permissions** file give the system administrator, who is working on the remote computer specified in the MACHINE entry, unrestricted access to the local system for **uucico** and **uuxqt** operations.

In addition to the unrestricted entry in the local **Systems** file, you can also set up a alternate **uucp** login in that file intended for users of remote systems whose access to the local system should be restricted. You then set up a LOGNAME entry in the local **Permissions** file that limits **uucico** and **uuxqt** activities on the local system. These activities have been requested by users working on the remote system identified with the alternate **uucp** login.

- For examples of such LOGNAME and MACHINE entries, see "Entries in the Systems and Permissions Files on a Local System" on page 9-61 and "Entries in the Systems and Permissions Files on the Remote Systems" on page 9-62.
- For additional examples of LOGNAME and MACHINE entries, see "Sample Configuration Files" on page 9-64.

Creating BNU Login IDs and Passwords

Following are instructions for creating a **uucp** login ID and its associated password. You issue the AIX command **users** for this task, and then follow the instructions on the screen.

If you are including two versions of the **uucp** program login ID in a **Permissions** file, first issue **users** and set up the unrestricted login and password. Then issue the command again and set up the restricted login and its associated password.

Note: The instructions below are very similar to those given in "Setting Up the BNU Manager's Login and Password" on page 9-17 for setting up a BNU system administrator login, with a few exceptions. You must be a member of the **system** group, or be operating with superuser authority, to execute the **users** command.

1. Enter the **users** command at the superuser prompt (#).
2. Enter the **add** subcommand to add the appropriate version of the **uucp** login ID:

   `> a u uucp`

   or

   `> a u uucpl`

3. Answer `n` to the `OK? (y)` prompt, and then enter the following in response to the `Field?` prompt:

   `Field? uid`

4. Now enter `5` in response to the prompt for the user ID:

   `uid 5`

5. Enter `group` in response to the next prompt:

   `Field? gr`

6. In response to the `group` prompt, enter the following:

   `group uucp`

7. Answer `n` to the `OK?` prompt, and then enter the following to set up a password for the new login ID:

   `Field? pa`

8. Enter the password for the login ID that you specified in step 2 in response to the `password` prompt.
9. The directory field should contain the following entry:

   `/usr/spool/uucppublic`

   If it does not, enter `dir` in response to the `Field?` prompt, and then enter the name of the BNU public directory in response to the prompt for a directory name.
10. The program field should contain the following entry:

    `/usr/lib/uucp/uucico`

    If it doesn't, enter that program name in response to the prompt.
11. To end the procedure, press **Enter** at the next `Field?` prompt. The **users** command displays the information for the new login ID (the password is encrypted).
12. If the information is correct, press **Enter** at the `OK? (y)` prompt and then press **Enter** again at the prompt `Standard new user initialization? (y)`. If the information is not yet correct, answer `n` to the `OK?` prompt and then re-enter the data.
13. Enter the **q** command to exit from the **users** program.

The system adds the new **uucp** login ID and password to the list of users in the **/etc/passwd** file, and the login ID to the **uucp::5:** entry in the **/etc/group** file.

Entries in the Systems and Permissions Files on a Local System

Following are example entries in the **Systems** file on the local system `zeus` for communications with the remote systems `hera`, `venus`, and `merlin`.

- System `zeus` is used only by the system manager. No other user at the site has a valid login on this computer. The login ID this system uses to initiate **uucico** and **uuxqt** activities on remote computers is `uucp`, and the associated password is `alluucp`.
- System `hera` is used only by members of the **system** group. Regular users cannot log in on this computer. System `hera` also uses the **uucp** login ID to initiate file-transfer activities, but the associated password is `sysuucp`.

- Both systems venus and merlin are available to all users working at the site. The login ID for these systems is uucp1, and the associated password is passuucp.

1. Following are the relevant entries in the **Systems** file on system zeus:

   ```
   hera Any hera 1200 - "" \r\d\r\d\r in:--in: uucp word: sysuucp

   venus Any venus 1200 - "" \r\d\r\d\r in:--in: uucp1 word: passuucp

   merlin Any merlin 1200 - "" \r\d\r\d\r in:--in: uucp1 word: passuucp
   ```

   Zeus can communicate with the three remote systems at any time. The local system sends the strings uucp and sysuucp in response to hera's login and password prompts, and the strings uucp1 and passuucp in response to the login and password prompts of venus and merlin.

2. Zeus is now configured to communicate with each of the other systems. However, before users on hera, venus, and merlin can log in to and use zeus for **uucico** and **uuxqt** activities, the following entries must exist in the **Permissions** file on zeus.

   ```
   LOGNAME=uucp REQUEST=yes SENDFILES=yes VALIDATE=hera READ=/ WRITE=/
   MACHINE=zeus:hera REQUEST=yes COMMANDS=ALL READ=/ WRITE=/

   LOGNAME=uucp1
   MACHINE=zeus:venus:merlin
   ```

   The first LOGNAME/MACHINE entry allows system hera, used only by the **system** group, free access to the files on zeus. The second LOGNAME/MACHINE entry allows systems venus and merlin only to send files to zeus.

Entries in the Systems and Permissions Files on the Remote Systems

The **Systems** file on the remote system hera contains the following entries for communicating with zeus, venus, and merlin:

```
zeus Any zeus 1200 - "" \r\d\r\d\r in:--in: uucp word: alluucp

venus Any venus 1200 - "" \r\d\r\d\r in:--in: uucp1 word: passuucp

merlin Any merlin 1200 - "" \r\d\r\d\r in:--in: uucp1 word: passuucp
```

Like zeus, system hera can communicate with the remote systems at any time. In response to zeus's login and password prompts, hera sends the strings uucp and alluucp, and the strings uucp1 and passuucp in response to the prompts from venus and merlin.

1. Following are the entries for the three remote systems in the **Systems** file on venus:

```
zeus 0800-1700 zeus 1200 - "" \r\d\r\d\r in:--in: uucp word: alluucp

hera 0800-1700 hera 1200 - "" \r\d\r\d\r in:--in: uucp word: sysuucp

merlin Any merlin 1200 - "" \r\d\r\d\r in:--in: uucp1 word: passuucp
```

2. The entries in the **Systems** file on merlin are almost exactly like those on venus:

```
zeus 0800-1700 zeus 1200 - "" \r\d\r\d\r in:--in: uucp word: alluucp

hera 0800-1700 hera 1200 - "" \r\d\r\d\r in:--in: uucp word: sysuucp

venus Any venus 1200 - "" \r\d\r\d\r in:--in: uucp1 word: passuucp
```

3. Following are the relevant entries in the **Permissions** file on hera for communications with the three other systems:

```
LOGNAME=uucp REQUEST=yes SENDFILES=yes VALIDATE=zeus READ=/ WRITE=/
MACHINE=hera:zeus REQUEST=yes COMMANDS=ALL READ=/ WRITE=/

LOGNAME=uucp1
MACHINE=hera:venus:merlin
```

The first LOGNAME/MACHINE entry allows zeus unlimited access to hera. The second entry allows venus and merlin only to send files to hera.

4. The entries in the **Permissions** file on venus contain the following permissions:

```
LOGNAME=uucp REQUEST=yes SENDFILES=yes VALIDATE=zeus:hera READ=/ WRITE=/
MACHINE=venus:zeus:hera REQUEST=yes COMMANDS=ALL READ=/ WRITE=/

LOGNAME=uucp1 REQUEST=yes SENDFILES=yes VALIDATE=merlin READ=/ WRITE=/
MACHINE=venus:merlin REQUEST=yes COMMANDS=ALL READ=/ WRITE=/
```

5. Again, the entries in the **Permissions** file on `merlin` are almost exactly like those on `venus`:

```
LOGNAME=uucp REQUEST=yes SENDFILES=yes VALIDATE=zeus:hera READ=/ WRITE=/
MACHINE=merlin:zeus:hera REQUEST=yes COMMANDS=ALL READ=/ WRITE=/

LOGNAME=uucp1 REQUEST=yes SENDFILES=yes VALIDATE=merlin READ=/ WRITE=/
MACHINE=merlin:venus REQUEST=yes COMMANDS=ALL READ=/ WRITE=/
```

For additional examples of entries in a **Permissions** file, see "Sample Configuration Files."

Sample Configuration Files

This section contains two sets of sample configuration files. Each example illustrates the entries in the relevant files that you need to include on both the local and the remote systems in order for the two computers to communicate using BNU.

Configuration Files for a Hardwired Connection

These files are set up for a hardwired connection between systems `zeus` and `hera`, where `zeus` is considered the local system and `hera` is considered the remote system. The hardwired device is `tty0`. The login ID for `zeus` is `uucp`, and the associated password is `alluucp`. The login ID for `hera` is also `uucp`, but the associated password is `sysuucp`.

Entries in the Local System's Files for a Hardwired Connection

1. The **Devices** file on `zeus` contains the following entry in order to connect to the remote system `hera`:

```
Direct  tty0  -  1200  direct
hera    tty0  -  1200  direct
```

2. The **Systems** file on `zeus` contains the following entry for the remote system `hera`:

```
hera Any hera 1200 - "" \r\d\r\d\r in:--in: uucp word: sysuucp
```

3. The **Permissions** file on the local system `zeus` contains the following entries specifying the ways in which the remote system `hera` can conduct **uucico** and **uuxqt** transactions with `zeus`:

```
LOGNAME=uucp VALIDATE=hera REQUEST=yes SENDFILES=yes READ=/ WRITE=/
MACHINE=zeus:hera REQUEST=yes SENDFILES=yes COMMANDS=ALL READ=/ WRITE=/
```

Entries in the Remote System's Files for a Hardwired Connection

1. The **Devices** file on hera contains the following entry for communications with zeus:

```
Direct  tty0  -  1200  direct
zeus    tty0  -  1200  direct
```

2. The **Systems** file on hera contains the following entry for zeus:

```
zeus Any zeus 1200 - "" \r\d\r\d\r in:--in: uucp word: alluucp
```

3. The **Permissions** file on hera contains the following entries specifying the ways in which zeus can conduct **uucico** and **uuxqt** transactions with hera:

```
LOGNAME=uucp VALIDATE=zeus REQUEST=yes SENDFILES=yes READ=/ WRITE=/
MACHINE=hera:zeus REQUEST=yes SENDFILES=yes COMMANDS=ALL READ=/ WRITE=/
```

Configuration Files for a Telephone Connection

These files are set up to connect systems venus and merlin over a telephone line using modems. Venus is considered the local system, and merlin is considered the remote system.

On both systems, the device tty1 is hooked to a Hayes modem at 1200 baud. The login ID for both systems is uucp1, and the associated password is passuucp. The phone number for the modem attached to venus is 9=3251436; the number of the merlin modem is 9=4458784. Both computers include partial phone numbers in their **Systems** files and dialing codes in their **Dialcodes** files.

Entries in the Local System's Files for a Telephone Connection

1. The **Devices** file on venus contains the following entry for the connection to merlin:

```
ACU  tty1  -  1200  hayes
```

2. The **Dialers** file on venus contains the following entry for its modem:

```
hayes  =,-,  ""  \dAT\r\c OK \pATDT\T\r\c CONNECT
```

3. The **Systems** file on venus contains the following entry for merlin, including a phone number and a dialing prefix:

```
merlin Any ACU 1200 local8784 "" \r\d\r\d\r in:--in: uucp1 word: passuucp
```

4. The **Dialcodes** file on venus contains the following dialing code prefix for use with the number in the **Systems** file:

```
local 9=445
```

5. The **Permissions** file on venus contains the following entries specifying the ways in which merlin can conduct **uucico** and **uuxqt** transactions with venus:

```
LOGNAME=uucp1 VALIDATE=merlin REQUEST=yes SENDFILES=yes READ=/ WRITE=/
MACHINE=venus:merlin REQUEST=yes SENDFILES=yes COMMANDS=ALL READ=/ WRITE=/
```

Entries in the Remote System's Files for a Telephone Connection

1. The **Devices** file on merlin contains the following entry for the connection to venus:

   ```
   ACU  tty1  -  1200  hayes
   ```

2. The **Dialers** file on merlin contains the following entry for its modem:

   ```
   hayes  =,-,  ""  \dAT\r\c OK \pATDT\T\r\c CONNECT
   ```

3. The **Systems** file on merlin contains the following entry for venus, including a phone number and a dialing prefix:

   ```
   venus Any ACU 1200 local436 "" \r\d\r\d\r in:--in: uucp1 word: passuucp
   ```

4. The **Dialcodes** file on merlin contains the following dialing code prefix for use with the number in the **Systems** file:

   ```
   local 9=325
   ```

5. The **Permissions** file on merlin contains the following entries specifying the ways in which venus can conduct **uucico** and **uuxqt** transactions with merlin:

   ```
   LOGNAME=uucp1 VALIDATE=venus REQUEST=yes SENDFILES=yes READ=/ WRITE=/
   MACHINE=merlin:venus REQUEST=yes SENDFILES=yes COMMAND=ALL READ=/ WRITE=/
   ```

For information about the hardware used in BNU communications, see "Managing Ports, Cables, and Modems" on page 5-39.

Checking for Correct Permissions

Now that the BNU files are customized for your site, you should issue the **uucheck** command again to check for possible errors in the **Permissions** file. However, **uucheck** does *not* check file/directory modes, nor does it check for duplicate login or machine names.

Issue **uucheck** with the **-v** flag, which provides a detailed explanation of the way in which BNU interprets the **Permissions** file:

```
uucheck -v
```

If **uucheck** displays an error message, use the **pg** command to examine the **Permissions** file and make sure the entries are correct. Then issue **uucheck** again.

When the shell prompt returns to the screen, continue with the next section.

Checking Networked Systems (uuname)

Before you continue, it's a good idea to check to see that all the RTs included in the **Systems** file on the local system are actually on the BNU network. Use the **uuname** command for this task.

```
uuname
```

If all the systems are networked correctly, each system name will appear on the list displayed on the screen. The RTs on this list are also the systems to which users can send mail.

Setting Up BNU for Use with TCP/IP

If your site is using TCP/IP, you will need to perform some additional tasks in order for BNU and TCP/IP to communicate. You must start the **uucpd** daemon, and you must include the relevant TCP/IP entries in the **Devices**, **Systems**, and **Permissions** files.

The **uucpd** daemon handles communications between BNU and TCP/IP. This daemon is an internal program that enables users on systems linked by BNU to establish a TCP/IP connection to other systems linked over a Token Ring or Ethernet network.

Following are the steps you need to perform to enable BNU and TCP/IP to communicate:

1. Check to see whether the **uucpd** daemon is running. If it is running, go on to step 5.
2. If the **uucpd** daemon is not running, enter the following command to start it manually as a background process:

   ```
   /usr/lib/uucp/uucpd &
   ```

3. Now make sure the **uucpd** daemon will run automatically the next time you start up the system by uncommenting the `uucpd` lines in the file **/etc/rc.tcpip**.
4. Check to see whether the file **/etc/services** includes the following line:

   ```
   uucp          540/tcp          uucpd
   ```

   If it doesn't, add that line.
5. Check to make sure that the host name table in the file **/etc/hosts** includes the name of the remote computer with which you want to connect.

   Note: Even if you have defined an alias for the host name in the domain name daemon's data base, you *cannot* use that alias in the **hosts** file (see the *Interface Program for use with TCP/IP*).
6. Update the **Systems**, **Devices**, and **Permissions** files in **/usr/adm/uucp** directory to include the relevant TCP/IP entries.

Even though the **uucpd** daemon is running, you must still add the relevant TCP/IP entries to the BNU configuration files. In order to make the appropriate entries in the **Systems** file, you must decide on the appropriate TCP/IP conversation protocol to enter in the `TCP` caller subfield.

There are three kinds of protocols: **g**, **t**, and **e**.

Protocol **Explanation**

g This is the default. The **g** protocol is useful for modem connections, but it involves a large overhead in running BNU commands.

t The **t** protocol presumes an error-free channel, and is essentially the **g** protocol without the checksumming and packetizing. You can use the **t** protocol for two types of communications:

- To communicate with a site running the AIX version of BNU
- To communicate with a site running the Berkeley version of the UNIX-to-UNIX Copy Program (UUCP).

The **t** protocol is not reliable for modem connections.

e You can also use the **e** protocol for two types of communications:

- To communicate with a site running the AIX version of BNU
- To communicate with a site running a non-AIX version of BNU.

The **e** protocol is not reliable for modem connections.

If you have not already done so, add the appropriate entries to the BNU configuration files.

- Enter the following in the **Devices** file:

  ```
  TCP  -  -  -  TCP  \D
  ```

  Specify `TCP` in the caller field. Enter hyphens in the line, line2, and class fields. Enter `TCP` as the dialer and either `\D` or `\T` as the token. Remember that the `\D` string instructs BNU to use the telephone number in the **Systems** file, but not to expand that number by including the dial code abbreviation specified in the **Dialcodes** file. If you want to expand the number with the dialing code abbreviation, use the characters `\T` instead of `\D`.

- Enter the following in the **Systems** file to use TCP/IP to connect to system `venus` with the default **g** protocol:

  ```
  venus  Any  TCP  -  -  in:--in: uucp1  word: passuucp
  ```

  Replace the send and expect characters in the sample login field with the login prompt, login, password prompt, and password appropriate to the remote system for which you are establishing a connection.

- Enter the following in the **Systems** file to use TCP/IP with the **t** protocol:

```
venus  Any  TCP,t  -  -  in:--in: uucp1  word: passuucp
```

  Again, replace the strings in the login field with the characters appropriate for your systems.

- Enter the following in the **Systems** file to use TCP/IP with the **e** protocol:

```
venus  Any  TCP,e  -  -  in:--in: uucp1  word: passuucp
```

  Again, replace the strings in the login field with the characters appropriate for your systems.

- Enter the following in the **Systems** file to use TCP/IP with a program login ID other than **uucp**:

```
opus  Any  TCP,t  -  -  in:--in: commo  word: bloom
```

- Enter the appropriate LOGNAME and MACHINE entries in the **Permissions** file:

```
LOGNAME=uucpo
MACHINE=zeus:venus:opus
```

  Depending on the remote systems that are connecting to the local system over the TCP/IP connection, use either the strict default permissions, or enter the options appropriate to your site.

Remember that you must set up an appropriate login ID and password for any remote system that will initiate **uucico** and **uuxqt** activities. For information about this task, see "Setting Up BNU Login IDs and Passwords" on page 9-59.

Performing Routine Maintenance Tasks

Once BNU is installed and configured to run at your site, you will have to perform some maintenance operations to ensure that the networking utilities run properly. Managing a small installation is relatively easy; managing a large BNU site is a little more difficult for several reasons, including the following:

- If there is a problem that requires making a change in one of the BNU configuration files, you must check and, if necessary, update all related entries in all relevant files on all the systems connected through BNU.
- Users, who are impatient with any system failure, are generally even less patient when failures occur between two systems that are under local control.

In general, the major problem in a network such as BNU is dealing with the backlog of file-transfer and command-execution requests that, for whatever reason, cannot be transmitted to the specified system. In order to minimize problems of this type, you should perform the tasks described in this section as part of your basic system maintenance procedures.

This section describes the following maintenance tasks:

- Monitoring remote connections and file transfers
- Cleaning up spooling directories
- Working with log files.

You must perform some parts of these tasks manually; BNU performs other parts automatically. The BNU software also contains several daemons and shell scripts that automatically execute certain maintenance operations, as described in "Using the Daemons" on page 9-82 and "Running Automatic Maintenance Routines" on page 9-97.

As you known, the **cron** daemon is very useful in managing AIX systems. When the local system is in run state 2 (multi-user), **cron** scans the **/usr/spool/cron/crontabs/uucp** file every minute for entries containing jobs scheduled for execution at that time. The automatic BNU maintenance routines use this feature of **cron** to help administer the networking facility.

Monitoring Remote Connections and File Transfers

BNU has several programs that enable you to monitor the progress of file transfers. These include the command **uustat**, which is employed by both general users and system administrators, and the command **Uutry**, which is used primarily by system administrators (although you do not need superuser privileges to issue **Uutry**).

The uustat Command

The BNU command **uustat** provides information about the status of several types of networking operations, including jobs that have been queued on a local system for transfer to a remote system. You will probably issue **uustat** frequently to check on the status of such queued jobs. The command is particularly helpful in monitoring requests for file transfers generated with the **uucp** and **uuto** commands, and requests for command executions generated with the **uux** command. (The **uustat** report also includes information about the status of mail activities.)

The **uustat** command also gives users limited control over BNU jobs queued to run on remote systems. By issuing the command with the appropriate flag, you can check the general status of BNU connections to other systems, and cancel transfer requests.

- For detailed information about the **uustat** command and numerous usage examples, see the BNU chapter in *Using the AIX Operating System.*
- For additional information about and examples of using **uustat**, see *AIX Operating System Commands Reference.*

The Uutry Command

The BNU command **Uutry**, which contacts a remote system with debugging turned on, is also extremely useful in monitoring the progress of a file transfer. In addition, the command enables you to monitor BNU connections to remote computers. **Uutry** (note the uppercase "U") invokes the **uucico** daemon, which is the BNU file-transport program, and then generates information about the way in which **uucico** is working.

You contact a specific remote system with **Uutry**, which produces debugging output that enables you to monitor the progress of **uucico** as it establishes the connection to the remote system, performs the remote login, and transfers a file. In addition to scrolling the debugging output on the screen of your local system, **Uutry** also directs this information to a file named /**tmp**/*system_name*, where *system_name* is the name of your local system.

The **Uutry** command has the following two flags:

-x*debug_level*

-r*system_name*

Debugging Level (-x*debug_level*)

The **Uutry** command produces a moderate amount of debugging information if you enter it without a flag. The command has a default ***debugging level*** of 5. (The debugging level, a single-digit number between 0 and 9, specifies the amount of debugging information that BNU places in the **tmp** file.) Higher numbers produce more detailed output, so the default produces a moderate amount of debugging information.

If you want more detailed information about the progress of the **uucico** operation, use the **-x** flag to specify a higher debugging level. For example,

```
Uutry -x9
```

instructs BNU to generate as much information as possible about the way in which **uucico** is working.

Override Retry Time (**-r***system_name*)

When you issue the **uucp**, **uuto**, or **uux** command, that command calls the **uucico** daemon, which contacts the remote system and transfers the file. (The BNU scheduler, **uusched**, also starts **uucico** automatically at specified intervals.) If for some reason **uucico** cannot complete the connection and transfer the file, the daemon waits for a set amount of time and then tries again. The default retry time is five minutes.

Note: The time the remote system was last polled is specified in the file **/usr/spool/uucp/.Status/***system_name*. However, the format of this file makes it somewhat difficult to read. To produce a more readable version, issue the **uustat -m** command.

When you invoke **uucico** with the **Uutry** command in order to monitor a remote connection or observe a file-transfer operation, you probably do not want to wait for several minutes between attempts to contact the remote system. In that case, use the **-r** flag to override the default retry time specified in the **.Status** file.

- For information about transporting files, see "Transporting Copy Requests" on page 9-82 and the BNU chapter in *Using the AIX Operating System.*
- For additional information about **uucico**, see "The uucico Daemon" on page 9-85 and the description of the command in *AIX Operating System Commands Reference.*
- For information about **uusched**, see "The uusched Daemon" on page 9-95 and the description of the command in *AIX Operating System Commands Reference.*

Monitoring a Remote Connection

As described above, the **uucico** daemon is generally invoked either by a command such as **uucp** or **uux**, or automatically by the **uusched** daemon. (Of course, you can also issue **uucico** manually.) The **Uutry** command, which is actually a shell script stored in the **/usr/adm/uucp** directory, enables you to invoke **uucico** directly from your local computer with debugging turned on. You can then verify that the specified remote system is up and running, or use the command to monitor the file-transfer process.

For **Uutry** to execute, you must either be in the **/usr/adm/uucp** directory when you issue the command, or you must include the complete path name to the shell script on the **Uutry** command line. If you use **Uutry** frequently, you can put the path name to the command in the PATH entry in your **.profile** file.

Monitoring a Successful Connection

Following is an example of the way in which **Uutry** can help you monitor the **uucico** process if users at your site report file-transfer problems.

1. First, issue **uustat** to determine the status of all the transfer jobs in the current queue:

   ```
   uustat -q
   ```

   The system displays a status report like the following:

   ```
   venus   3C  (2)  05/09-11:02  CAN'T ACCESS DEVICE
   hera    1C       05/09-11:12  SUCCESSFUL
   merlin  2C       05/09-10:54  NO DEVICES AVAILABLE
   ```

   This report indicates that three command (**C.***) files intended for system `venus` have been in the queue for two days. There could be several reasons for this delay—for example, `venus` could have been shut down for maintenance, the modem could be turned off, and so on.

2. Before you begin more extensive troubleshooting activities, issue **Uutry** to determine whether your local system can contact `venus` at this time:

   ```
   /usr/adm/uucp/Uutry -r venus
   ```

   This command starts **uucico** with a moderate amount of debugging and the instruction to override the default retry time. The **Uutry** command directs the debugging output to a temporary file, `/tmp/venus`, and executes a **tail -f** command.

   Note: If there is a large amount of debugging information, you can use the **INTERRUPT** sequence (**Alt-Pause** on the RT) to return to the **Uutry** shell while the command continues to place the debugging output in the temporary file. Then, when the operation is finished, issue the **pg** command to examine the `/tmp/venus` file.

3. If your local system succeeds in establishing a connection to `venus`, the debugging output will contain a good deal of information. However, the final line in this script is the most important:

   ```
   Conversation Complete:  Status SUCCEEDED
   ```

4. In the case of a successful connection, assume that the temporary file-transfer problems are now resolved, and issue **uustat** again to make certain that the files in the spooling directory were transferred successfully to `venus`.

Monitoring an Unsuccessful Connection

1. Issue **uustat** to determine the status of the jobs intended for the remote system.
2. If the status report indicates a problem transferring a file to a specific remote system, issue the **Uutry** command to determine whether your local system can now contact that remote system.
3. If your local system cannot contact the remote system `venus`, the debugging output generated by **Uutry** will contain the following type of information (the exact form of the output may vary):

   ```
   mchFind called (venus)
   conn (venus)
   getto ret -1
   Call Failed:  CAN'T ACCESS DEVICE
   exit code 101
   Conversation Complete:  Status FAILED
   ```

4. First check the physical connections between the local and the remote systems. Make sure that the remote computer is turned on and that all the cables are properly connected, that the ports are enabled or disabled (as appropriate) on both systems, that the modems (if applicable) are working, and so on.

   For detailed information about this hardware, see "Managing Ports, Cables, and Modems" on page 5-39.

5. If the physical connections are correct and secure, then you must check all the relevant configuration files on both the local and the remote systems.
   - Make certain that the entries in the **Devices**, **Systems**, and **Permissions** files are correct on both systems.
   - In addition, if you are using a modem, make sure that the **Dialers** and **Dialcodes** files contain the proper entries.
   - If you are using a TCP/IP connection, make sure the **uucpd** daemon is running, and that the configuration files contain the correct `TCP` entries.
6. Once you have checked the physical connections and configuration files, issue **Uutry** again. If the debugging output still reports that the connection failed, you may need to confer with a member of your systems support team. Be sure to save the debugging output produced by the **Uutry** command; it may prove helpful in diagnosing the problem.

Monitoring a File Transfer

In addition to monitoring remote connections, the **Uutry** command also enables you to monitor file transfers. The two processes are very similar.

1. Issue the **uustat** command to check the status of the files in the spooling directory on your local system.
2. Now issue **Uutry** to verify that the local system can contact the remote system.
3. If the debugging output indicates that the connection was not successful, follow the steps described in "Monitoring an Unsuccessful Connection" on page 9-74. If the connection was successful, go on to step 4.
4. Now prepare a file for transfer using the **uucp** command with the **-r** flag:

   ```
   uucp -r test1 venus!~/test2
   ```

   The **-r** option instructs BNU to place the `test1` file in the queue, but *not* to start the **uucico** daemon.
5. Now issue **Uutry** with the **-r** flag to start **uucico** with debugging turned on. The daemon contacts `venus`, logs in, and transfers the file, while **Uutry** produces debugging output that enables you to monitor the **uucico** process.

For additional information about handling file-transport problems, see "Login Failures" on page 9-103.

Cleaning Up the Spooling Directories

Each system connected by BNU has several types of spooling directories:

- The **/usr/spool/uucp/***system_name* directory contains queued requests issued by local users for file transfers to remote systems, and for command executions on remote systems. The queued files remain in this spooling directory until they are transferred to the designated system, removed manually with the **uucleanup** command, or removed automatically by the **uudemon.cleanu** shell script.
- The **/usr/spool/uucppublic** directory is the BNU public directory. When a user transfers a file to a remote system, or issues a request to execute a command on another system, the files generated by these BNU commands are stored in the public directory on the designated system. (However, the user can specify a destination other than the public directory when issuing the **uucp**, **uuto**, or **uux** commands.) Again, the transferred files remain in this directory until they are removed manually or automatically.

This section includes information about the **uucleanup** command and about cleaning up the files in the spooling and public directories.

The uucleanup Command

The **uucleanup** program scans the **/usr/spool/uucp/***system_name* directory for files older than a specified period of time and takes appropriate action to remove such files in a useful way. The command does the following:

- Informs the system manager of requests to send files to and receive files from remote systems that the local system cannot contact.
- Warns users about requests that have been waiting in the spooling directory for a given period of time (the default is one day).
- Returns to the original sender mail that cannot be delivered to the requested recipient.
- Removes all other files older than a specified number of days from the spooling directory.

Note: Depending on the size of your installation and the available storage space on the local system, you can set the default for any length of time, but you should probably allow files to remain in the spooling directory for at least the default number of days, as described in the command options, below.

The **uucleanup** program is started by the shell **uudemon.cleanu**, located in **/usr/adm/uucp**. The shell script is, in turn, started periodically by the **cron** daemon, based on instructions in the file **/usr/spool/cron/crontabs/uucp**. In general, **uucleanup** is executed automatically, although you can start it manually if you are logged in as **su**.

Note: Automatic cleanup is not enabled when BNU is installed. As described in "Cleaning Up Files and Directories (uudemon.cleanu)" on page 9-98, you must edit the file **/usr/spool/cron/crontabs/uucp** and remove the comment character (#) from the beginning of the **uudemon.cleanu** line to enable the automatic cleanup routine.

For additional information about automatic cleanup procedures, see the following:

- "Cleaning Up Undeliverable Jobs" on page 9-78
- "Cleaning Up the Public Directory" on page 9-79
- "Running Automatic Maintenance Routines" on page 9-97.

Following is the form of the **uucleanup** command:

uucleanup [*options*]

The command includes the following flags:

-C*time*	**-m***string*
-D*time*	**-o***time*
-W*time*	**-s***system*
-X*time*	

Remove Command Files (-C*time*)

This flag removes any command (**C.***) files as old as, or older than, the number of days specified in *time*, and sends notice of the removal to the user who requested the file transfer or command execution. Unless you specify otherwise, the default *time* is seven days.

For information about command files, see "Additional Information about Command Files" on page 9-86.

Remove Data Files (-D*time*)

This flag removes any data (**D.***) files as old as, or older than, the number of days specified in *time*. Again, the default is seven days. This option also instructs **uucleanup** to attempt to deliver any mail messages in the spooling directory.

For information about data files, see "Additional Information about Data Files" on page 9-89.

Warn Requestor about Remaining Command Files (-W*time*)

This flag sends a mail message to the user who requested the file transfer or command execution, warning that command files as old as, or older than the number of days specified in *time* are still in the spooling directory. The default is one day. The message includes the job ID and, in the case of mail, the actual mail message. You can use the **-m** option to include a message line telling the user who to contact to check on the transfer problem.

Remove Execute Files (-X*time*)

This flag removes any execute (**X.***) files as old as, or older than, the number of days specified in *time*. The default is two days. There are probably no data files related to these execute files. If any related data files do exist, however, the **-D***time* processing removes them, as described above.

For information about execute files, see "Additional Information about Execute Files" on page 9-92.

Send Message Line (-m*string*)

This flag instructs the **uucleanup** command to include a specified line of text in the warning message generated by the **-W***time* option. The default message line is "See your local administrator to locate the problem."

Remove Other Files (-o*time*)

This flag removes files other than command, data, or execute files. These other files must be as old as, or older than the number of days specified in *time*. The default is two days.

Execute Only on Specified System (-s*system*)

This flag executes the **uucleanup** command only on the spooling directory on the specified system. The default is to clean up all the BNU spooling directories.

Note: Unless you set one of the *time* flags to a specific number of days, **uucleanup** uses the default time values.

Cleaning Up Undeliverable Jobs

You should issue the **uustat** command frequently to determine the status of the connections to various remote systems, and to check the number and age of the file-transfer and command execution requests queued in the **/usr/spool/uucp/***system_name* directory.

Note: You should also uncomment the **uudemon.admin** shell script line in the **/usr/spool/cron/crontabs/uucp** file so that you automatically receive status information about the queue at least once a day. For information about enabling this feature, see "Getting Status Information (uudemon.admin)" on page 9-97.

The status information you need to check carefully includes the following :

- The age in days of the oldest request in each queue
- The number of times the local system has tried and failed to reach the specified computer
- The reason for the failure to contact the specified system.

You can probably delete (either manually or automatically) execute files that have been in the queue for at least two days. The reason they are still queued is that the related data files required to execute the specified command on the designated system were not transferred, for some reason. Since **D.*** files are generally sent at the same time as **X.*** files, the transfer probably failed at the point of destination.

For information about monitoring file-transfer failures, see "Monitoring a File Transfer" on page 9-75 and "Login Failures" on page 9-103.

You should be careful, however, before deleting old command files created to transfer files from the local to a remote system. Before removing these old **C.*** files, you should make every possible effort to establish the connection and transfer the files. Again, see "Monitoring a File Transfer" on page 9-75 for information about monitoring remote-system connections and file-transfer requests.

As described above, you can remove these files manually, by issuing the **uucleanup** command with the appropriate options, or you can have them removed automatically by the **uudemon.cleanu** shell script, which is described in some detail in "Cleaning Up Files and Directories (uudemon.cleanu)" on page 9-98.

Cleaning Up the Public Directory

All spooling directories are dynamic, including the public directory. Depending upon the size of your installation and the number of files sent to the local **/usr/spool/uucppublic** directory by users on remote systems, this directory may at times become quite large.

The **uudemon.cleanu** shell script cleans up all the BNU spooling directories, including the public directories, unless you direct it to clean up only the directories on one specific system by issuing the the **uucleanup -s***system_name* command.

In order to keep the local file system from overflowing when users send files to the public directory, the **uudemon.cleanu** script includes a **find** command intended to deal with this problem. The **find** command locates any empty directories and any files older than seven days, and the shell script then deletes these items.

If the local system does not have enough storage space to permit a large **/usr/spool/uucppublic** directory, you can change the seven-day default to a shorter time period.

Working with Log Files

BNU creates individual log files for each remote system with which your local system communicates using the **uucp**, **uuto**, or **uux** commands.

BNU normally places status information about each transaction in the appropriate log file each time you use the networking utilities facility. When more than one BNU process is running, however, the system cannot access the log file, so it places this status information in a file with a **.LOG** prefix that covers just the single transaction.

You can use **uulog** to display a summary of **uucp** and **uux** requests by user or by system. All these transactions are logged in files with a form of the name **/usr/spool/uucp/.LOG/***/daemon_name/system_name.*

- The **uucp** and **uuto** commands are executed by the **uucico** daemon. The **uucico** activities are logged in **/usr/spool/uucp/.LOG/uucico/***system_name.*
- The **uux** command is executed by the **uuxqt** daemon. The **uuxqt** activities are logged in **/usr/spool/uucp/.LOG/uuxqt/***system_name.*

You can examine these individual log files by issuing **uulog** directly. However, you can also have BNU automatically append these temporary log files to a primary log file. This is called ***compacting*** the log files, and it is handled by the **uudemon.cleanu** shell script, as described in "Compacting Log Files" on page 9-80.

The uulog Command

Following is the form of the **uulog** command:

uulog [*options*]

The command includes the following flags:

-f*system_name*	**-x**
-s*system_name*	-*number*

Display Log for Specified System (-f*system_name*)

This flag performs a **tail -f** on the file-transfer log for the specified system. In this case, that means BNU displays the end of the log file. Use **INTERRUPT** (**Alt-Pause** on the RT) to leave the file and return to the prompt.

Display Summary of Transfer Requests (-s*system_name*)

This flag displays a summary of information about **uucico** activities for the specified system.

Display uuxqt Log (-x*system_name*)

This flag displays the **uuxqt** log file for the specified system.

Execute tail -f for Specified Lines (-*number*)

This flag instructs BNU to execute a **tail -f** command for the specified number of lines.

Compacting Log Files

As discussed above, you can use **uulog** to access the individual log files on each computer. For example, system `hera` has a log file for **uucico** requests and a log file for **uuxqt** requests, and you can examine each of these files with **uulog**.

However, you may find it more useful to have BNU automatically append these various log files to one primary log file, rather than examining each file individually. The **uudemon.cleanu** shell script handles this task.

As discussed in "Running Automatic Maintenance Routines" on page 9-97, the instructions governing the execution of the automatic maintenance routines are contained in the **/usr/spool/cron/crontabs/uucp** file. The **cron** daemon automatically runs commands and shell scripts at specified dates and times according to the instructions you include in this **crontabs/uucp** file.

Whenever **cron** executes **uudemon.cleanu**, that script combines the **uucico** and **uuxqt** log files on a system and stores them in a directory named **/usr/spool/uucp/.Old**. The default is for **uudemon.cleanu** to save log files that are two days old.

You can change the default by simply modifying the appropriate line in the shell script. If storage space is a problem on a particular system, consider reducing the number of days that the files are kept in the individual log files.

Cleaning Up sulog and cron/log

The **/usr/adm/sulog** and **/usr/adm/cron/log** files are both related indirectly to BNU transactions.

The **sulog** file contains a history of superuser (**su**) command usage. Because the various **uudemon** entries in the **/usr/spool/cron/crontabs/uucp** file each use the **su** command, the **sulog** file can grow quite large over a period of time. You should purge this file periodically to keep it to a reasonable size.

In a similar way, the **/usr/adm/cron/log** file contains a history of all the processes generated by **/etc/cron**. Over a period of time, this file too will become quite large, and you should purge it periodically to limit its size.

Using the Daemons

BNU contains two daemons that handle file transfers and remote-command executions. These are the **uucico** daemon, which transports files from one system to another, and the **uuxqt** daemon, which executes specified AIX commands on designated systems.

In addition to these daemons, BNU also includes a daemon that schedules the transfer of files queued in the spooling directory. This is the **uusched** daemon.

This section includes the following types of information:

- How BNU transfers copy requests, using the **uucp** command and the **uucico** daemon
- How BNU executes remote commands, using the **uux** command and the **uuxqt** daemon
- How BNU schedules work in the spooling directory.

Transporting Copy Requests

BNU is used primarily to perform these kinds of tasks:

- Transfer files within and between systems, using the **uucp** and **uuto** commands and the **uucico** daemon
- Request a remote system to run AIX commands for another system, using the **uux** command and the **uuxqt** daemon. For information about running AIX commands on remote systems, see "Executing Remote Commands" on page 9-90.

Following is a brief overview of the file-transfer process:

1. A user issues the BNU command **uucp** (or **uuto**) to send a source file from a local to a remote system. This command carries out the file transfer in two steps: It first creates a command file in the spooling directory on the local computer, and then calls **uucico** to send the source and command files to the remote computer.
2. If the user has invoked **uucp** with the **-C** flag to send the file to the spooling directory for transfer, **uucp** creates not only a command file, but also a data file that contains the actual source file.
3. Once the command file (and data file, if necessary) is created, **uucp** calls **uucico**, which in turn contacts the remote computer and delivers the file.

The rest of this section contains the following information:

- An overview of the directories and files used in transferring files and executing command remotely
- A brief discussion of the **uucp** command

- A discussion of the **uucico** daemon
- Examples of command and data files.

Overview of Files and Directories

BNU uses a number of directories and files when transferring information and running commands remotely. Following are brief descriptions of some of these directories and files:

The /usr/adm/uucp Directory

This directory contains the files in the supporting data base, several versions of the **uutry** command, and four scripts for automatic maintenance routines.

The /usr/lib/uucp Directory

This directory contains the **uucheck** and **uucleanup** commands, and the **uucico**, **uucpd**, **uusched**, and **uuxqt** daemons.

The /usr/spool/uucp/.Admin Directory

This directory contains the following files:

audit	**Foreign**
errors	**xferstats**

The most important of these is the **xferstats** file. This file includes information about the status of the transfer request, including such items as the system name and the name of the user requesting the transfer, the date and time of the transfer, the name of the device used in the transfer, the size of the transferred file, and so on.

The /usr/spool/uucp/.Corrupt Directory

This directory contains copies of files that, for some reason, could not be processed. If, for example, a file is not in the correct form for transfer, BNU places a copy of that file in the **.Corrupt** directory for later handling by the system manager. However, this directory is rarely used.

The /usr/spool/uucppublic Directory

This is the public directory for the BNU facility, and one of these directories exists on every system connected by the networking utilities. When a user transfers a file to a remote system, or issues a request to execute a command on another system, the files generated by these BNU commands are stored in the public directory on the designated system until the destination directory is ready to receive them. (A user can also specify a destination other than the public directory when issuing the **uucp**, **uuto**, or **uux** commands.)

The /usr/spool/uucp/.Status Directory

This directory contains additional types of information about the file transfer, such as the machine called, the time of the last call in seconds, the status of the last call, the number of retrys, the retry time in seconds to the next call, and so on.

The /usr/spool/uucp/*system_name* Directory

This is the spooling directory on the local system. It contains queued requests issued by local users for file transfers to remote systems and for command executions on remote system.

The /usr/spool/uucp/Xqtdir Directory

This directory contains execute files with lists of commands that remote system may run.

The /usr/spool/uucp/.Workspace Directory

This directory holds temporary files of various kinds that the file transport programs use internally.

BNU also uses three types of administrative files to transfer data between systems:

Command/work files	These files contain the directions for **uucico**.
Data files	These files contain the data to be sent to remote systems.
Execute files	These files contain instructions for running commands that require the resources of a remote system.

- For detailed information about command files, see "Additional Information about Command Files" on page 9-86.
- For detailed information about data files, see "Additional Information about Data Files" on page 9-89
- For detailed information about execute files, see "Additional Information about Execute Files" on page 9-92.

The uucp Command

Users issue the **uucp** command to copy one or more source files from one AIX system to one or more destination files on another AIX system.

The **uucp** command first creates a command file in the spooling directory on the local system. If the user has issued the command with the **-C** flag, **uucp** also creates a data file that contains the actual source file. After creating the command file (and data file, if necessary), **uucp** calls the **uucico** daemon, which handles the actual file transfer.

- For detailed information about using the **uucp** command, see the BNU chapter in *Using the AIX Operating System.*
- For detailed information about the command itself, see the description of **uucp** in *AIX Operating System Commands Reference.*

The uucico Daemon

The **uucico** daemon selects the device to be used for the communications link, establishes the connection with the specified system, invokes the appropriate login sequence, checks permissions, transfers the command (and data) files, logs the results of the transfer, and notifies a designated user that the transfer is complete. Both the local and the remote systems run **uucico**, and the two daemons "talk" to each other to perform these tasks.

Specifically, **uucico** assists **uucp** (and **uux**) by performing the following actions:

- Scanning the spooling directory on the local system for transfer requests
- Placing a call to the specified remote system
- Negotiating a line protocol to be used to connect the local and remote systems
- Running all transfer requests from both the local and the remote system
- Logging transfer requests and completions.

Note: In the case of a **uux** request for the execution of an AIX command on a remote system, **uucico** transfers the files, and then the **uuxqt** daemon executes the command on the remote system. For information about **uuxqt**, see "Executing Remote Commands" on page 9-90.

When a user issues **uucp** or **uux**, those commands call **uucico**, which runs as a background process. In addition, **uucico** is also started periodically by the BNU scheduler, **uusched**. The scheduling daemon is, in turn, started periodically by the **cron** daemon, based on instructions in the **/usr/spool/cron/crontabs/uucp** file. These instructions execute the shell script **/usr/adm/uucp/uudemon.hour**.

- For detailed information about **uusched**, see "Scheduling Work in the Spooling Directory" on page 9-95.
- For detailed information about the **uudemon.hour** shell script, see "Calling File-transport Programs (uudemon.hour)" on page 9-99.

You can also start **uucico** manually for debugging purposes. Following is the form of the command:

uucico [*options*]

Note: You must either be in the **/usr/lib/uucp** directory when you issue the **uucico** command, or you must enter the command with the full path name: **/usr/lib/uucp/uucico**.

The **uucico** command uses the following flags:

-r*role_number*
-s*system_name*
-x*debug_level*

Role Number (**-r***role_number*)

This flag specifies the server and client relationship. The role numbers are 1 for the server mode and 0 for the client mode. The default is 0 because **uucico** is generally started automatically by a program such as **uucp** or by the **cron** daemon. If you start **uucico** manually, set this flag to 1.

Remote System Name (**-s***system_name*)

This flag specifies the name of the remote system. Use this flag only when you are starting **uucico** manually. BNU supplies the system name internally when **uucico** is started automatically.

Debug Level (**-x***debug_level*)

This flag produces debugging information about the progress of the **uucico** activity. The valid range for the debugging level is 0 to 9, with a default of 5. Higher numbers produce more detailed debugging information.

Note: As described in "Monitoring a File Transfer" on page 9-75, the BNU command **Uutry** also starts **uucico** with debugging turned on.

When starting **uucico** manually, use the following form to run the daemon as a background process on the computer specified by **-s***system_name*:

`/usr/lib/uucp/uucico -r1 -s`*system_name* `&`

- For additional information about using the **uucico** daemon for debugging, and for example debugging output, see "Login Failures" on page 9-103.

Additional Information about Command Files

The full path name of a command/work file is a form of the following:

/usr/spool/uucp/*system_name***/C.***system_nameNxxxx*

where the first system name is that of the local system and the name following the **C.** notation is that of the remote computer, the *N* character represents the grade of the work, and the *xxxx* notation is the four-digit hexadecimal transfer-sequence number—for example, `C.merlinC3119`.

Note: The grade of the work specifies when the file is to be transmitted during a particular connection. The grade notation is a single number (0-9) or letter (A-Z, a-z). Lower sequence characters cause the file to be transmitted earlier in the connection than do higher sequence characters. The number 0 is the highest grade, signifying the earliest transmittal; z is the lowest grade, specifying the latest transmittal. The default grade is **N**.

A command file consists of a single line that includes the following kinds of information in the following order:

1. An **S** (send) or **R** (receive) notation.

   Note: A "send" command file is created by the **uucp** or **uuto** commands; a "receive" command file is created by the **uux** command.

2. The full path name of the source file being transferred. A receive command file, discussed below, does not include this entry.
3. The full path name of the destination file, or a path name preceded by ~*user*, where *user* is a login name on the specified system.
4. The sender's login name.
5. A list of the options, if any, included with the **uucp**, **uuto**, or **uux** command.
6. The name of the data file associated with the command file in the spooling directory. This field must contain an entry. If one of the data-transfer commands (such as **uucp** with the default **-c** flag) does not create a data file, BNU instead creates a placeholder with the name **D.0** for send files, or **dummy** for receive files.
7. The source file's permissions code, specified as a three-digit octal number (for example, 777).
8. The login name of the user on the remote system who is to be notified when the transfer is complete.

Examples of Two Send Command Files

The send command file `/usr/spool/uucp/venus/C.heraN1133`, created with the **uucp** command, contains the following fields:

```
S /u/amy/f1 /usr/spool/uucppublic/f2 amy -dC D.herale73655 777 lgh
```

where

1. `S` denotes that **uucp** is sending the file.
2. The full path name of the source file is `/u/amy/f1`.
3. The full path name of the destination is `/usr/spool/uucppublic/f2`, where `/usr/spool/uucppublic` is the name of the BNU public spooling directory on the remote computer and `f2` is the new name of the file.

   Note: The destination name may be abbreviated as `~uucp/f2`. The tilde (~) is a shorthand way of designating the public directory.

4. The person sending the file is `amy`.

5. The sender entered the **uucp** command with the `-C` option, specifying that **uucp** should transfer the file to the local spooling directory and create a data file for it. (The `-d` option, which specifies that the command should create any intermediate directories needed to copy the source file to the destination, is a default.)
6. The name of the data file, `D.herale73655`, which **uucp** assigns.
7. The octal permissions code `777`.
8. The login name of the user on `hera` who is to be notified of the file's arrival, `lgh`.

Look at another send command file, `/usr/spool/uucp/hera/C.zeusN3130`. In this case, the user issued the **uuto** command to produce the following file:

```
S  /u/amy/out  ~/receive/msg/zeus  amy  -dcn  D.0  777  msg
```

The `S` denotes that the source file `/u/amy/out` was sent to the `receive/msg` subdirectory in the public spooling directory on system `zeus` by user `amy`.

Note: The **uuto** command creates the directory `receive/msg` if it does not already exist.

The **uuto** command used the defaults `-d` (create directories), `-c` (transfer directly, no spooling directory or data file), and `-n` (notify recipient). The `D.0` notation is a placeholder, `777` is the permissions code, and `msg` is the recipient.

Example of a Receive Command File

The format of a receive command file is somewhat different from that of a send command file. The **uux** command creates a receive command file when files required to run a specified AIX command on a remote system are not present on that specified system.

For example, the command

```
uux - "diff /u/amy/out hera!/u/amy/out2 > ~uucp/DF"
```

produces a receive command file named `/usr/spool/uucp/zeus/C.heraR1e94`.

Note: The command in this example invokes `uux` to run a `diff` command on the local system, comparing file `/u/amy/out` with file `/u/amy/out2`, which is stored on remote system `hera`. The output of the comparison is placed in file `DF` in the public directory on the local system.

The actual receive command file looks like this:

```
R  /u/amy/out2 D.herale954fd  amy  -  dummy  0666  amy
```

The `R` denotes a receive file. The **uucico** daemon, called by `uux`, gets the file `/u/amy/out2` from `hera` and places it in a data file called `D.herale954fd` for the transfer. Once the files are transferred, the **uuxqt** daemon executes the command on the specified system.

User amy issued the uux command with the - (minus sign) option, which makes the standard input to uux the standard input to the actual command string. No data file was created in the local spooling directory, so BNU uses dummy as a placeholder. The permissions code is 666 (BNU prefixes the three-digit octal code with a 0), and user amy is to be notified when the command has finished executing.

Additional Information about Data Files

The full path name of a data file is a form of the following:

/usr/spool/uucp/*system_name***/D.***system_namexxxx###*

where the first system name is that of the local system and the name following the **D.** notation is the name of the remote computer. The *xxxx###* notation is the hexadecimal sequence number of the command file associated with that data file—for example, D.venus471afd8.

After a set period of time (specified in the **uusched** program), the **uucico** daemon transfers the data file to the designated system. It places the original data file in a subdirectory of the BNU spooling directory named **/usr/spool/uucp/***system_name*, where *system_name* is the name of the computer that is transmitting the file. BNU creates a temporary data file to hold the original data file.

The full path name of the temporary data file is a form of the following:

/usr/spool/uucp/*system_name/TM.00PID.000*

where *system_name* is the name of the computer that is sending the file, and *TM.xxPID.000* is the name of the file—for example, TM.00451.000. (The PID number is the process ID of the job.)

After receiving the entire file, BNU takes one of three actions:

- If the file was sent with **uucp** and there were no transfer problems, the program immediately renames the **TM.*** file with the appropriate data file name, such as D.venus471afd8, and sends it to the specified destination.
- If the file was sent with the **uuto** command, BNU also renames the temporary data file with the appropriate **D.*** name. It then places the data file in the public directory, **/usr/spool/uucppublic**, where the user receives and handles it with one of the **uupick** options.
- If there were transfer problems (such as a failed login or an unavailable device), the temporary data file remains in the spooling subdirectory. The **uudemon.cleanu** shell script removes these files automatically at specified intervals, or you can remove them manually.

Executing Remote Commands

Users can run AIX commands on remote systems by invoking the BNU command **uux**, which functions in the following way:

1. The command gathers any necessary files from the designated systems.
2. It then creates a command file, a data file, and an execute file. The command files contain the same type of information as those created by **uucp**, while **uux** data files either contain the data for a remote command execution, or else become execute files on remote systems for remote command execution. Execute files contain the command string to be run on the specified system.
3. After creating the files, **uux** calls the **uucico** daemon, which contacts the designated system and delivers the files.
4. The **uuxqt** daemon executes the command on the specified system, placing any output from the command in a designated file on a specified system.

In executing remote commands, BNU uses the same directories and files described in "Overview of Files and Directories" on page 9-83. However, in addition to command and data files, **uux** also requires execute **(X.*)** files, which are described in detail in "Additional Information about Execute Files" on page 9-92.

The uux Command

As described above, the **uux** command runs a specified AIX command on a specified AIX system.

After creating the required command, data, and execute files, **uux** calls the **uucico** daemon to transfer the files from the spooling directory on the local system to the designated remote system. Once the files are transferred, the **uuxqt** daemon executes the command string contained in the execute file on the remote system.

The command string is made up of one or more arguments that look like an AIX command line, except that the command string may be prefixed by the name of the remote system in the form *system_name*!. Unless the user entering the **uux** command includes the **-n** flag, the command notifies that user if the remote system does not run the command. This response comes by mail from the remote system.

- For detailed information about using the **uux** command, refer to the BNU chapter in *Using the AIX Operating System.*
- For detailed information about the command itself, see the description of **uux** in *AIX Operating System Commands Reference.*

The uuxqt Daemon

After a user enters **uux** to execute an AIX command on a designated system, the BNU command generates command, data, and execute files. The execute file, described in detail in "Additional Information about Execute Files" on page 9-92, contains the names of the files needed to run the command on the remote computer. The **uux** command then calls the **uucico** daemon.

The **uucico** daemon places the transferred files in the spooling directory on the designated system. The **uuxqt** daemon searches this directory periodically for **X.*** files whose names indicate that they have been sent from another system. When it finds such a file, **uuxqt** takes the following actions:

- Reads the file to determine the list of data files required for the command execution
- Checks to see whether the required data files are available and accessible
- Checks the **Permissions** file to verify that the requested command may be executed on that system.

You can start **uuxqt** manually, but the daemon is generally started automatically at specified intervals by the **uudemon.hour** shell script, which is stored in **/usr/adm/uucp**. The shell script, in turn, is started periodically by the **cron** daemon, based on instructions in the file **/usr/spool/cron/crontabs/uucp**.

For information about the **uudemon.hour** shell script, see "Calling File-transport Programs (uudemon.hour)" on page 9-99.

You start the **uuxqt** daemon manually by issuing the following command:

uuxqt [*options*]

Note: You must either be in the **/usr/lib/uucp** directory when you issue the **uuxqt** command, or you must enter the full path name of the command: **/usr/lib/uucp/uuxqt**.

This command uses the following flags:

-s*system_name*
-x*debug_level*

Remote System Name (-s*system_name*)

This flag specifies the name of the remote system. Use this flag only when you are starting **uuxqt** manually. BNU supplies the system name internally when **uuxqt** is started automatically.

Debug Level (-x*debug_level*)

This flag produces debugging information about the progress of the **uuxqt** activity. The valid range for the debugging level is 0 to 9, with a default of 5. Higher numbers produce more detailed debugging information.

When starting **uucico** manually, use the following form to run the daemon as a background process on the computer specified by **-s***system_name*:

```
/usr/lib/uucp/uuxqt -ssystem_name &
```

Limiting the Number of Remote Executions

The **Maxuuxqts** file, located in the **/usr/adm/uucp** directory, limits the number of **uuxqt** processes running simultaneously on a local system.

This file contains an ASCII number that you can change in order to meet the needs of your installation; the default is 2. In general, the larger the number, the greater the potential load on the local system.

The **Maxuuxqts** file requires no configuration and no maintenance unless the system on which it is installed is utilized frequently and heavily by users on remote systems.

Additional Information about Execute Files

The full path name of a **uux** execute file is a form of the following:

/usr/spool/uucp/*system_name***/X.***system_nameNxxxx*

where the system name preceding the **X.** notation is the name of the local computer and the name following it is the name of the remote system. The *N* character represents the grade of the work, and the *xxxx* notation is the four-digit hexadecimal transfer-sequence number—for example, `X.zeusN2121`.

Note: Notice that the only difference between the name of a command file and that of an execute file is the **C.** or **X.** following the name of the local system. As was also the case with command files, the grade of an execute file specifies when that file is to be transmitted during a particular connection. Lower sequence characters cause the file to be transmitted earlier than do higher sequence characters. The default grade is **N**.

Standard Entries in an Execute File

An execute file consists of several lines, each with an identification character and one or more entries:

User Line

U *user_name system_name*

This user line includes the login name of the user issuing the **uux** command, and the system from which the command was issued.

Error Status Line

N or Z

The **N** character means that a failure message is *not* sent to the user issuing **uux** if the specified AIX command does not execute successfully on the remote system.

The **Z** character means that a failure message is sent to the user issuing **uux** if the specified AIX command does not execute successfully on the remote system.

Requestor's Name

R *user_name*

This is the login ID of the user requesting the remote command execution.

Required File Line

F *file_name*

This line contains the names of the files required to execute the specified command on the remote system. The *file_name* can be either the complete path name of the file including the unique transmission name assigned by BNU, or simply the transmission name without any path information (see the examples below). This line can contain zero or more file names. The **uuxqt** daemon checks for the existence of all listed files before running the specified AIX command.

Standard Input Line

I *file_name*

The standard input is either specified by a < (less than) symbol in the command string, or inherited from the standard input of the **uux** command if that command was issued with the hyphen (-) flag.

If standard input is specified, it also appears in an F (Required File) line. If standard input is not specified, BNU uses **/dev/null**.

Standard Output Line

O *file_name system_name*

This is the name of the file and system that are to receive standard output from the command execution. Standard output is specified by a > (greater than) symbol within the command string. (You cannot use the > > sequence in **uux** commands.) As was the case with standard input, if standard output is not specified, BNU uses **/dev/null**.

Command Line

C *command_string*

This is the command string that the user requests to be run on the specified system. BNU checks the **Permissions** file on the designated computer to see whether the login ID may run the AIX command on that system.

All required files go to the execute directory, usually **/usr/spool/uucp/.Xqtdir**. After execution, the standard output is sent to the requested location.

Sample Execute Files

User amy on local system zeus issued the following command:

```
uux - "diff /u/amy/out hera!/u/amy/out2 > ~uucp/DF"
```

The command in this example invokes uux to run a diff command on the local system, comparing file /u/amy/out with file /u/amy/out2, which is stored on remote system hera. The output of the comparison is placed in file DF in the public directory on the local system.

That command produced an execute file named /usr/spool/uucp/hera/X.zeusN212F, which contains the following information:

```
U amy zeus
# return status on failure
Z
# return address for status or input return
R amy
F /usr/spool/uucp/hera/D.hera1e954fd out2
O ~uucp/DF zeus
C diff /u/amy/out out2
```

The user line identifies amy on zeus. The error status line indicates that amy will receive a failure status message if the diff command fails to execute. The requestor is amy, and the file required to execute the command is the following data file:

```
/usr/spool/uucp/hera/D.hera1e954fd out2
```

The output of the command is to be written to the public directory on zeus with the name DF. (Remember that ~uucp is the shorthand way of specifying the public directory.) The final line is the command string that amy entered with the **uux** command.

Following is another example of an execute file:

```
U uucp hera
# don't return status on failure
N
# return address for status or input return
R uucp
F D.hera5eb7f7b
I D.hera5eb7f7b
C rmail amy
```

This indicates that user uucp on system hera is sending mail to user amy, who is also working on system hera.

Scheduling Work in the Spooling Directory

When users issue BNU commands to copy files and execute remote commands, the files containing these work requests are queued for transfer in the local **/usr/spool/uucp/***system_name* directory. The BNU daemon **uusched** schedules the transfer of these files.

This section contains information about the **uusched** daemon, and about the way in which the **Maxuuscheds** file works with that daemon to limit the number of remote systems that **uucico** can contact any one time.

The uusched Daemon

This program schedules the transfer of files that are queued in the spooling directory, **/usr/spool/uucp**, on the local system.

The **uusched** daemon first randomizes the queued work in the spooling directory and then starts the **uucico** daemon with the **-s***system_name* option. This flag tells **uucico** the system for which the selected job is scheduled.

You can start **uusched** manually, but the daemon is generally started automatically at specified intervals by the **uudemon.hour** shell script, which is stored in **/usr/adm/uucp**. The shell script, in turn, is started periodically by the **cron** daemon, based on instructions in the file **/usr/spool/cron/crontabs/uucp**.

For information about the **uudemon.hour** shell script, see "Calling File-transport Programs (uudemon.hour)" on page 9-99.

You start the **uusched** daemon manually by issuing the following command:

uusched [*options*]

Note: You must either be in the **/usr/lib/uucp** directory when you issue the **uusched** command, or you must enter the full path name of the command: **/usr/lib/uucp/uusched**.

This command uses the following flags:

-u*debug_level*
-x*debug_level*

Pass Debug Level to uucico (-u*debug_level***)**

This flag passes the **-x***debug_level* specification on to the **uucico** daemon, which then produces debugging output about the file-transport activities.

Debug Level (-x*debug_level***)**

This flag produces debugging information about the progress of the **uusched** activity. The valid range for the debugging level is 0 to 9, with a default of 5. Higher numbers produce more detailed debugging information.

When starting **uusched** manually, use the following form to run the daemon as a background process:

```
/usr/lib/uucp/uusched &
```

Limiting the Number of Scheduled Jobs

The file **/usr/adm/uucp/Maxuuscheds** limits the number of remote systems that the **uucico** program can contact at any one time. This file is used in conjunction with the **uusched** daemon and the lock files in the **/etc/locks** directory to determine the number of systems currently being polled. You use this file to help manage system resources and load averages.

The **Maxuuscheds** file contains an ASCII number that you can change in order to meet the needs of your installation; the default is 2. In general, the larger the number, the greater the potential load on the local system.

The **Maxuuscheds** file requires no configuration and no maintenance unless the system on which it is installed is utilized frequently and heavily by users on remote systems.

Running Automatic Maintenance Routines

The BNU software includes four shell scripts that run maintenance routines automatically. These include the following:

uudemon.admin	**uudemon.hour**
uudemon.cleanu	**uudemon.poll**

These shell scripts reside in the directory **/usr/adm/uucp**. They are started periodically by the **cron** daemon, based on instructions in the file **/usr/spool/cron/crontabs/uucp**.

The automatic maintenance routines are not enabled when you install BNU. You must edit the **crontabs/uucp** file, removing the comment character (#) from the beginning of each individual line that governs running the particular maintenance shell script, as described in the following sections.

Getting Status Information (uudemon.admin)

The **uudemon.admin** shell script mails status information about the BNU activities to the **uucp** login ID at intervals specified in the **crontabs/uucp** file. The shell script executes both the **uustat -p** and the **uustat -q** commands.

- The **-p** flag instructs **uustat** to run a **ps -flp** (process status: full, long list of specified process IDs) for all PID numbers in the lock files.
- The **-q** flag lists the jobs currently queued to run on each system; these jobs are either waiting to execute or in the process of executing. If a status file exists for the system, its date, time, and status information are reported.

You should execute the **uudemon.admin** shell script at least once a day. To run it automatically, uncomment the following line in the **/usr/spool/cron/crontabs/uucp** file by deleting the pound sign (#) that precedes this line:

```
48 8,12,16 * * * /bin/sh "/usr/adm/uucp/uudemon.admin > /dev/null"
```

The 48 notation represents minutes, the 8,12,16 notation represents hours based on the 24-hour clock, and the three asterisks (* * *) are placeholders representing the day of the month, the month of the year, and the day of the week. This line therefore instructs the **cron** daemon to run the **uudemon.admin** shell script at 48 minutes past the hours 8, 12, and 16—that is, at 8:48 a.m., 12:48 p.m., and 4:48 p.m.

Note: Remember that these run intervals are just defaults. You can change the times at which **cron** executes **uudemon.admin** to fit the needs of your site simply by changing these time intervals.

Use the **pg** command to examine the **uudemon.admin** script, which is located in the directory **usr/adm/uucp**.

Cleaning Up Files and Directories (uudemon.cleanu)

As described in "Cleaning Up the Spooling Directories" on page 9-75, the **uudemon.cleanu** shell script cleans up the BNU spooling directories and log files. The script deletes files in the spooling directory that are as old as, or older than a specified number of days, and removes empty directories.

The **uudemon.cleanu** shell script also updates archived log files, removing log information more than three days old. The script removes log files for individual computers from the **usr/spool/uucp/.Log** directory, merges them, and places them in the directory **usr/spool/uucp/.Old**, which contains old log information.

After performing the clean-up operations, the script mails the **uucp** login ID a summary of the status information gathered during the current day.

You can instruct **cron** to run the **uudemon.cleanu** shell script daily, weekly, or at longer intervals, depending on the amount of **uucico** and **uuxqt** transactions that occur on the local system.

To run this script automatically, uncomment the following line in the **/usr/spool/cron/crontabs/uucp** file by deleting the pound sign (#) that precedes this line:

```
45 23 * * * /bin/sh "/usr/adm/uucp/uudemon.cleanu > /dev/null"
```

The 45 notation represents minutes, the 23 notation represents hours based on the 24-hour clock, and the three asterisks (* * *) are placeholders representing the day of the month, the month of the year, and the day of the week. This line therefore instructs **cron** to run the **uudemon.cleanu** shell script at 45 minutes after hour 23—that is, at 11:45 p.m.

Remember that these run intervals are just defaults. You can change the times at which **cron** executes **uudemon.cleanu** to fit the needs of your site simply by changing these time intervals.

Note: AIX allots BNU a specified amount of storage space for any one particular log file; the number of blocks is determined by the default **ulimit**. If the **uudemon.cleanu** script fails to execute because the **ulimit** is set too low for the requirements of the local system, delete the line shown above from the **crontabs/uucp** file and add the following entry to the root **crontabs** file:

```
45 23 * * * ulimit 5000; /bin/su uucp -c "/usr/adm/uucp/uudemon.cleanu
> /dev/null"
```

You can use the **pg** command to examine the **uudemon.uucleanup** script, which is located in the directory **usr/adm/uucp**.

Calling File-transport Programs (uudemon.hour)

The **uudemon.hour** shell script is used in conjunction with the **Poll** file, the **uudemon.poll** shell script, and the **/usr/spool/cron/crontabs/uucp** file to initiate calls to remote systems.

Specifically, **uudemon.hour** calls various programs involved in transferring files between systems at specified hourly intervals. It calls the following:

- The **uusched** daemon, which first searches the spooling directory on the local system for command files that have not been transferred to the specified remote system, and then schedules the transfer of those files
- The **uuxqt** daemon, which searches the spooling directory for execute files that have been transferred to the local system but not yet processed on that system.

You can instruct **cron** to run the **uudemon.hour** shell script at specified hourly intervals. The frequency at which you run the script depends entirely on the amount of file-transfer activity originating from the local computer. If users on the local system initiate a large number of file transfers, you may need to specify that **cron** should start **uudemon.hour** several times an hour. If the number of file transfers originating from the local system is low, you can probably specify a start time once every four hours, for example.

To run this script automatically, uncomment the following line in the **/usr/spool/cron/crontabs/uucp** file by deleting the pound sign (#) that precedes this line:

```
25,55 * * * /bin/sh  "/usr/adm/uucp/uudemon.hour > /dev/null"
```

The `25,55` notation represents minutes, and the three asterisks (* * *) are placeholders representing the day of the month, the month of the year, and the day of the week. This line therefore instructs **cron** to run **uudemon.hour** at `25` minutes past the hour and again at `55` minutes past the hour—for example, at 8:25 and 8:55 a.m., 9:25 and 9:55 a.m., and so on.

Again, note that these run intervals are defaults, which you can change to meet the needs of your site simply by changing the time intervals. For example, to run the shell script once every four hours, enter the number `4` in the time-interval field.

You can use the **pg** command to examine the **uudemon.hour** script, which is located in the directory **usr/adm/uucp**.

Polling Remote Systems (uudemon.poll)

As mentioned above, the **uudemon.poll** shell script is used in conjunction with the **Poll** file, the **uudemon.hour** shell script, and the **/usr/spool/cron/crontabs/uucp** file to initiate calls to remote systems.

The **uudemon.poll** shell script performs the following actions:

- Polls the systems listed in the **Poll** file (**/usr/adm/uucp/Poll**)
- Creates command files for the systems listed in the **Poll** file.

The time at which you run **uudemon.poll** depends on the time at which you run **uudemon.hour**. You generally schedule the polling shell script to before the hourly script. This schedule enables **uudemon.poll** to create any required command files before **cron** runs **uudemon.hour**.

Instruct **cron** to run **uudemon.poll** about five to ten minutes before running **uudemon.hour**. To run this script automatically, uncomment the following line in the **/usr/spool/cron/crontabs/uucp** file by deleting the pound sign (#) that precedes this line:

```
20,50 * * * /bin/sh "/usr/adm/uucp/uudemon.poll > /dev/null"
```

As was the case with the **uudemon.hour** script, the `20,50` notation represents minutes, and the three asterisks (*** * ***) are placeholders representing the day of the month, the month of the year, and the day of the week. This line therefore instructs **cron** to run **uudemon.poll** at 20 minutes past the hour and again at 50 minutes past the hour—for example, at 8:20 and 8:50 a.m., 9:20 and 9:50 a.m., and so on.

Again, these run intervals are defaults. You can change the times at which **cron** executes **uudemon.poll** to correspond to the times you set up for **uudemon.hour**. The defaults specified in **/usr/spool/cron/crontabs/uucp** run **uudemon.poll** five minutes before running **uudemon.hour**.

You can use the **pg** command to examine the **uudemon.poll** script, which is located in the directory **usr/adm/uucp**.

Handling Common Problems

Much of the material in this chapter has dealt with handling problems that users may encounter when working with the BNU facility. This section deals briefly with the following situations:

- Full spooling directories
- Untransferred files
- Outdated **Systems** files
- Faulty automatic call units (ACUs) and modems
- Login failures.

Full Spooling Directories

If users at your site often transfer files and execute commands remotely using the BNU facility, the file system that spools incoming or outgoing work requests can run out of space. This makes it impossible to send jobs to, or receive jobs from remote systems.

The inability to receive files from remote systems is possibly the more serious of the two problems. When file space for the incoming work does become available, the local system may be flooded with a backlog of work sent from remote systems.

This type of problem is easy to solve. Simply modify the line in the **/usr/spool/cron/crontabs/uucp** file that starts the **uudemon.cleanu** shell script. Change the numbers in the first two fields of the cleanup line to decrease the intervals between the times when the **cron** daemon executes **uudemon.cleanu**.

For detailed information about cleaning up the spooling directories and modifying the cleanup shell script, see the following:

- "Cleaning Up the Spooling Directories" on page 9-75
- "Cleaning Up Files and Directories (uudemon.cleanu)" on page 9-98.

Untransferred Files

If you need to transfer files from a local to a remote system immediately, rather than waiting for the execute time specified in the **/usr/spool/cron/crontabs/uucp** file, you can manually issue either the **uucico** command or the **uuxqt** command.

Use the **uucico** daemon to manually transfer data (**D.***) files. Use the **uuxqt** daemon to manually transfer execute (**X.***) files.

Note: If you wish to monitor the process of the file transfer, do *not* issue either command with the ampersand (&) character, which instructs AIX to run the specified command as a background process.

For detailed information about these procedures, see the following:

- "Monitoring a File Transfer" on page 9-75
- "Transporting Copy Requests" on page 9-82.

Outdated Systems Files

It can sometimes be difficult to maintain current **Systems** files when the users at your site communicate regularly with users at a remote site.

This can be a costly problem if your site communicates with the remote computers over telephone lines. For example, a modem and line may be unavailable for a significant length of time while a local system attempts to contact a remote system. This is particularly frustrating if the user cannot reach the remote system simply because the phone number of its modem has been changed, or the system manager at the remote site has issued a new login ID or password for that system.

The best way to deal with this problem is to try to avoid it by communicating regularly with the system managers at remote sites. Be sure to contact the administrators of remote computers that are part of your network whenever you change a telephone number, a login ID, or a password—and request that these individuals send you the same information when they make similar changes to the configuration files of the computers at their installations.

For detailed information about the **Systems** file, see "Customizing the Systems File" on page 9-33.

Faulty Automatic Call Units (ACUs) and Modems

The ACUs used at your site, as well as the incoming modems, may occasionally cause problems that make it difficult for users to contact other computers or to receive files.

You can generally identify such problems relatively easily because the status report generated with the **uustat** command includes the number of times that BNU has tried to reach the designated system, and the reason why the contact failed.

If you suspect that the problem is caused by a bad transmission line, use the **cu** command to monitor the line. Issue **cu** to connect to a different system than the one listed in the status report as having problems, but specify the suspected line. If the second system also has trouble connecting over the specified line, check the hardware.

- For information about using the **cu** command, see the BNU chapter in *AIX Operating System Commands Reference*. For information about the command itself, see the **cu** entry in *AIX Operating System Commands Reference.*
- For information about the hardware involved in establishing a remote connection, see "Managing Ports, Cables, and Modems" on page 5-39.
- For information about ACUs, see "Setting Up Devices" on page 9-22.

If users on specific local systems at your site are having problems communicating with specific remote systems using modem connections, you should also make certain that the configuration files on all the involved systems are set up correctly. For information about configuring these files, see "Setting Up Remote Communications" on page 9-20.

Login Failures

Two of the most common reasons for login failures result from the following:

- The appropriate tty device has either not been created at all, or has not been created properly.
- The expect-send sequence in the login field in the **Systems** file needs additional data in order to establish a reliable connection to a certain remote system.

Problems with a tty Device

If the tty device has not been created, BNU will probably display the following error message when a user attempts to contact a remote system:

```
Call failed:  NO DEVICES AVAILABLE
```

If the tty device has not been properly created, BNU will probably display this error message:

```
Call failed:  CAN'T ACCESS DEVICE
```

You then need to perform one of the following actions:

1. If the device does not exist, create the appropriate tty device with the **devices** command.
2. If the device has not been created properly, check the value of the **dvam** parameter for that tty and, if necessary, change it with the **devices** command.
3. If the proper device exists, either the direct or the modem connection may be wrong. Depending on the configuration, check either or both of these connections, and then check the configuration files on both the local and the remote systems.

For detailed information about these operations, see the following:

- For information about the **devices** command, see the description of that command in *AIX Operating System Commands Reference.*
- For information about creating and managing the devices involved in remote communications, see "Managing Ports, Cables, and Modems" on page 5-39.
- For information about debugging connection problems, see "Monitoring a File Transfer" on page 9-75 and "Troubleshooting Connection Problems" on page 9-105.

Problems with an Expect-send Sequence

You may occasionally need to change the *expect-send* character strings in the *Login* field in the **Systems** file, depending on the way in which the local system or the remote system is initialized.

Note: If you are having problems that appear to be related to expect-send characters, you can manually start the **uucico** daemon with the **-x** (debug) flag to monitor the communication, as described in "Monitoring a File Transfer" on page 9-75 and "Troubleshooting Connection Problems" on page 9-105.

If you or your users experience login failures when trying to connect to a remote computer specified correctly in the **Systems** file, you may need to add several additional characters to the appropriate line in the entry representing that system. These additional characters precede the data included in the login field.

Suppose your **Systems** file contains the following entry:

```
hera  Any  hera  1200  -  in:--in: uucp  word: sysuucp
```

This specifies a hardwired connection to system `hera` (entered in both the system-name and caller fields) at a transmission rate of `1200` bps. The first expect string, `in:--in:`, contains the login prompt issued by `hera`, and the first send string contains the `uucp` program login ID sent by the local system. The second expect string contains `hera`'s password prompt, represented by `word`, and the second send string contains the local system's response, `sysuucp`.

If this entry is producing login failures for which you cannot account in any other way, try including one of the following standard strings in the relevant line in the **Systems** file:

1. `"" \n`
2. `"" \r\d\r\d\r`

Add one of the strings to the entry and see if that solves the problem. If it does not, delete that string, add the other string, and try again. Your entry in the **Systems** file will look like one of the following examples:

```
hera  Any  hera  1200  -  "" \n in:--in: uucp  word:  sysuucp
```

or

```
hera  Any  hera  1200  -  "" \r\d\r\d\r  in:--in: uucp  word: sysuucp
```

If you continue to have login problems, check the appropriate entries in the **Devices**, **Systems**, and **Permissions** files to make certain that the remote system is correctly specified.

For additional information about these files, see the following:

- "Setting Up Devices" on page 9-22
- "Customizing the Systems File" on page 9-33
- "Customizing the Permissions File" on page 9-43.

For information about debugging file-transfer and connection problems, see "Monitoring a File Transfer" on page 9-75 and "Troubleshooting Connection Problems."

Troubleshooting Connection Problems

As described in "Monitoring a File Transfer" on page 9-75, you use a form of the **uucico** command to debug a connection problem:

`/usr/lib/uucp/uucico -r1 -s`*`system_name`* `-x9`

where `-r1` specifies the server mode, `-s`*`system_name`* is the name of the remote system to which you are trying to connect, and `-x9` specifies the highest debug level, which produces the most detailed debugging information.

Expect-send debugging output produced by the **uucico** command can come either from information in the **Dialers** file, or from information in the **Systems** file.

Note: To set up a connection with a remote system, you must be familiar with the login sequence of that system. For detailed information about expect-send sequences, see "Expect-send Characters in Login Fields" on page 9-37.

Problems in the Dialers File

When establishing a connection between a local and a remote system using a telephone line and modem, BNU consults the **Dialers** file. (BNU also checks the **Systems** file to make sure it contains a listing for the specified remote computer.) Each entry in the **Dialers** file is a line that contains a series of expect-send sequences, like the following:

```
hayes   =,-,  ""  \dAT\r\c OK \pATDT\T\r\c CONNECT
```

This example specifies a Hayes modem:

- The `=,-` notation translates the telephone number: any `=` represents "wait for dial tone," and any `-` represents "pause."
- The `""` notation tells the modem to wait for nothing; continue with the rest of the string.
- The `\dAT` notation instructs the modem to delay for `AT` (the Hayes "Attention" prefix).

The remainder of the line in the **Dialers** file for the Hayes modem contains similar information. If the line in the **Dialers** file is not set up correctly for the specified modem, BNU will probably display the following error message:

```
DIALER SCRIPT FAILED
```

1. First, make sure that both the local and the remote modems are turned on, that they are both set up correctly, and that the telephone number of the remote modem is correct.
2. Then check the **Dialers** file and make sure the information is correctly specified for the local modem. If possible, you should also check the **Dialers** file on the remote system.
3. If you continue to have connection problems, check the documentation that came with your modem to make sure you have used the correct expect-send sequence characters in the **Dialers** file.

- For detailed information about the hardware used in establishing remote communications, see "Managing Ports, Cables, and Modems" on page 5-39.
- For detailed information about configuring the **Dialers** file, see "Setting Up Dialers" on page 9-30.

Problems in the Systems File

When establishing any type of connection, BNU consults the **Systems** file, which specifies the way in which the local computer communicates with a designated remote computer.

For example, suppose your local **Systems** file contains the following entry for communicating with the remote system `venus`:

```
venus  Any  venus  1200  -  "" \n in:--in: uucp1  word: passuucp
```

If users on the local system are having trouble connecting to `venus`, you can issue the **uucico** command with the debugging flag to produce debugging information about the connection:

```
/usr/lib/uucp/uucico -r1 -svenus -x9
```

Note: When entered with the appropriate flag, the **ct** and **cu** commands also produce debugging information about remote connections.

Following is an example of the debugging output produced when the **uucico** command (or **ct** or **cu**) uses the entry in the **Systems** file shown above to connect the local system to the remote system `venus`:

```
expect: ""
got it
sendthem (^J^M)
expect (in:)
^M^Jlogin:got it
sendthem (uucp1^M)
expect (word:)
^M^JPassword:got it
sendthem (passuucp^M)
imsg >^M^J^PShere^@Login Successful: System=venus
```

These entries represent the following information:

expect: ""	The local system should not wait for any information from the remote system.
got it	Acknowledgement.
sendthem (^J^M)	The local system sends the remote system a carriage return and a new line.
expect (in:)	The local system expects to receive the remote system login prompt, which ends in the character string `in:`.
^M^Jlogin:got it	The local system receives the remote login prompt.

sendthem (uucp1^M)	The local system sends the uucp1 login ID to the remote system.
expect (word:)	The local system expects to receive the remote system password prompt, which ends in the character string word:.
^M^JPassword:got it	The local system receives the remote password prompt.
sendthem (passuucp^M)	The local system sends the password for the uucp1 login ID to the remote system.
imsg >^M^J^PShere^@Login Successful: System=venus	The local system is successfully logged in to the remote system venus.

Chapter 10. Overview of the Message Handling Package

CONTENTS

About This Chapter

This chapter provides a brief description of the files, commands, and features of the MH package. This chapter also describes some of the managing tasks associated with using the MH package.

Understanding the MH Package

The MH (Message Handling) package contains a collection of commands that enable you to create, distribute, receive, view, process, and store messages and to verify user IDs, address formats, and date formats. You can run the Message Handling (MH) commands as separate programs, use the Message Handling (MH) commands with other Message Handling (MH) and AIX commands, and create programs that call Message Handling (MH) commands.

The MH package uses the AIX file and directory system. Message Handling (MH) commands store messages as individual files with numerical file names, and group messages into directories. A directory of messages is called a ***folder***. Each folder contains some standard files to support the Message Handling (MH) functions, as well as its actual messages. Since the messages are AIX files and the folders are AIX directories, you can use the AIX commands to manage messages and folders. For example, you can use the **chmod** command to change the access permissions on a folder, or you can run **cron** to purge old or inactive messages from a group of folders.

The MH package does not provide a message transport facility. Message Handling (MH) commands provide input to the **sendmail** command, and rely on the transport facilities associated with **sendmail**. See Chapter 7, "Managing the Electronic Mail System" on page 7-1 for more information.

Standard Directories

Files supporting the MH package are stored in several directories. The following paragraphs describe these directories and the types of Message Handling (MH) files that they contain:

Directory	Description
/usr/lib/mh	Contains files that describe the system's Message Handling (MH) defaults. You can alter many of these files to set your own system defaults.
$HOME	Contains the user's Message Handling (MH) profile *.mh_profile*. Users can modify their own *.mh_profile* to set their own Message Handling (MH) defaults.
user_mh_directory	Contains the user's message folders and usually the user's **context** file.

Files in the Supporting Database

The following files are used by the Message Handling (MH) system:

File	Description
$HOME/.mh_profile	The Message Handling (MH) user profile.
user_mh_directory/**draft**	The default draft file.
/usr/mail/$USER	The location of the user's mail drop.
/usr/lib/mh/maildelivery	The default local delivery instructions file.
$HOME/.maildelivery	The user's local delivery instructions file.
/usr/lib/mh/MailAliases	The default mail alias file.
/usr/lib/mh/mtstailor	The Message Handling (MH) tailor file.
/usr/lib/mh/components	The default message form for the **comp** command. Each user can create a default message form and store that form in the file *user_mh_directory*/**components**.
/usr/lib/mh/distcomps	The default message skeleton for the **dist** command. Each user can create a default message skeleton and store that message skeleton in the file *user_mh_directory*/**distcomps**.
/usr/lib/mh/forwcomps	The default message skeleton for the **forw** command. Each user can create a default message skeleton and store that message skeleton in the file *user_mh_directory*/**forwcomps**.
/usr/lib/mh/digestcomps	The default message skeleton for the **forw** command when the **-digest** flag is specified. Each user can create a default message skeleton for digests and store that message skeleton in the file *user_mh_directory*/**digestcomps**.
/usr/lib/mh/mhl.forward	The default message filter for the **forw** command when the **-format** flag is specified. Each user can create a default filter and store that filter in the file *user_mh_directory*/**mhl.forward**.
/usr/lib/mh/mhl.format	The default message template. Each user can create a default message template and store that template in the file *user_mh_directory*/**mhl.format**.
/usr/lib/mh/replcomps	The default reply template. Each user can create a default reply template and store that template in the file *user_mh_directory*/**replcomps**.
/tmp/prompter*	A temporary copy of a message. This file is created when the editor **prompter** is used.

User Commands

The following commands are the user-level programs in the MH package package:

Command	Description
ali	Lists mail aliases and their addresses.
anno	Annotates messages.
ap	Parses and reformats addresses.
burst	Explodes digests into messages.
comp	Composes a message.
dist	Redistributes a message to additional addresses.
dp	Parses and reformats addresses.
folder	Selects and lists folders and messages.
folders	Lists folders and messages.
forw	Forwards messages.
inc	Incorporates new mail.
mark	Creates, modifies, and displays message sequences.
mhl	Produces formatted listings of messages.
mhmail	Sends or receives mail.
mhpath	Prints full path names of messages and folders.
msgchk	Checks for messages.
msh	Creates a Message Handling (MH) shell.
next	Shows the next message.
packf	Compresses the contents of a folder into a file.
pick	Selects messages by content, and creates and modifies sequences.
prev	Shows the Previous message.
refile	Files messages in other folders.
repl	Replies to a message.
rmf	Removes a folder.
rmm	Removes messages.
scan	Produces a one line per message scan listing.
send	Sends a message.
show	Shows messages.
sortm	Sorts messages.
vmh	Invokes a visual interface for use with Message Handling (MH) commands.
whatnow	Invokes a prompting interface for draft disposition.
whom	Lists the addresses of the proposed recipients of a message and verifies the addresses.

AIX Operating System Commands Reference provides a detailed description for each of these Message Handling (MH) commands.

Other MH Commands

The following Message Handling (MH) commands can be be used by other programs, but should not be run directly by users:

Command	Description
conflict	Searches for alias and password conflicts.
install-mh	Initializes the Message Handling (MH) environment.
post	Delivers a message.
prompter	Invokes a prompting editor.
rcvdist	Sends a copy of incoming messages to additional recipients.
rcvpack	Saves incoming messages in a packed file.
rcvstore	Incorporates new mail from standard input into a folder.
rcvtty	Notifies the user of incoming messages.
slocal	Processes incoming mail.
spost	Delivers a message.

AIX Operating System Commands Reference provides a detailed description for each of these Message Handling (MH) commands.

Message Components

Each message contains a ***header*** and a ***body***. The header provides date, address, and subject information. The body provides the text of the message.

The ***header*** can contain various ***components***, or fields. These components may be generated by such sources as the MH package (for example, **From:**), the message transport system (for example, **Date:**), the sender (for example, **To:**), and the recipient (for example, **Replied:**). Each header component starts with the name of the component and is followed by a colon and the information in that component. If the component information requires more that one line, you must begin each continuation line with a space character. The first line of a component can not begin with a space character. The following list describes some of the header components:

Header Component	Description
To:	Lists the addresses or aliases of the primary recipients of the message.
cc:	Lists the addresses or aliases of the "carbon copy" recipients of the message.
Bcc:	Lists the addresses or aliases of the "blind carbon copy" recipients of the message. Although persons listed in this component receive a copy of the message, the MH package removes this component from the sent copy of the message.

Subject: Describes the subject or contents of the message.

Date: Reports the date the message was sent.

From: Reports the sender of the message.

The **mh-mail** section in *AIX Operating System Technical Reference* contains a more complete list of header components.

The ***body*** of a message consists of the text of the message. The body follows the header. While you compose a message, a line of dashes separates the header from the body. Do not remove this separator line. When you send the message, the MH package converts this line of dashes to an empty line.

Installing the MH Package

The MH package files are located on the AIX **Extended Services** diskette. Refer to *Installing and Customizing the AIX Operating System* for information about how to install Extended Services programs.

The first time each user runs a Message Handling (MH) command, that command calls **install-mh**, which creates a Message Handling (MH) profile for that user. See the **install-mh** section in *AIX Operating System Commands Reference* for more information about this command.

Performing Routine Maintenance Tasks

The MH package requires that you perform a few routine maintenance tasks. Since the MH package creates backup files, you will need to remove files periodically. You may also want to verify addresses and check alias files for duplicate aliases. The following sections discuss these maintenance tasks.

Removing Messages and Folders

The MH package provides you with two commands for removing messages and folders. The **rmm** command removes messages, and the **rmf** command removes folders.

The **rmm** command does not actually delete messages. It renames messages by placing a **,** (comma) in front of the message file names. You can still use AIX commands to manipulate these file, but the comma makes the messages unavailable to the MH package. Periodically, you should delete these files. You can place an entry in your **crontab** file and use the **crontab** command to automatically delete all files having file names beginning with a comma.

The **rmf** command removes folders, but does not remove the files within folders. Thus, before removing a folder, make sure all files within the folder are deleted.

Since messages are AIX files and folders are AIX directories, you can use AIX commands (such as **rmdir**, and **rm**, and **del**) to remove messages and folders.

See *AIX Operating System Commands Reference* for more information about the commands: **rmf**, **rmm**, **rmdir**, **rm**, **del**, and **crontab**.

Checking for Invalid Addresses

The Message Handling (MH) command **whom** expands address headers into sets of addresses and optionally verifies whether the addresses are valid. A valid address is an address that has a format acceptable to the mail transport system. Valid address headers do not imply that a message is deliverable. See the **whom** section in *AIX Operating System Commands Reference* for more information.

Checking for Duplicate Aliases and for Inappropriate Mail Drops

The Message Handling (MH) command **conflict** checks for duplicate aliases and for inappropriate mail drops. **conflict** is not a user command, but is designed to be run by other programs (such as **mhmail**). See the **conflict** section of *AIX Operating System Commands Reference* for more information.

Understanding MH Defaults

The MH package has a multilayered default structure. When you enter a Message Handling (MH) command, the system searches for arguments and default settings in the following sequence:

1. The system accepts all valid arguments and flags from the command line (or from wherever the command was initiated).
2. The system searches **$HOME/.mh_profile** for the entry *command:*, where *command* is the name by which the program was invoked. If this profile entry exists, the system accepts all arguments that do not cancel the previously stated command line instructions.

   Note: If you invoke a Message Handling (MH) command by specifying a link to the command, the system searches for the profile entry having that link as its name. If you invoke the command by specifying an alias, the system searches for the profile entry having the actual command as its name, rather than the alias as its name.
3. The system searches **$HOME/.mh_profile** for other entries that are applicable to the command and that have not already been overridden by previously stated arguments. If any such entries exist, the system accepts the arguments to those entries.
4. The system uses the current state information (such as the current folder and the current message) recorded in *user_mh_directory*/**context** for any arguments that have not been stated.
5. The system accepts system-wide defaults for any remaining unspecified arguments.

When you specify a relative path name for your Message Handling (MH) profile, the Message Handling (MH) programs interpret the full path name as beginning at the current directory. If you specify a relative path name for any other Message Handling (MH) files, the Message Handling (MH) programs interpret the full path names for those files as beginning at *user_mh_directory*.

Tailoring MH Profiles and Format Files

The MH package allows you to tailor Message Handling (MH) system files and user files. The following sections provide information about tailoring Message Handling (MH) files.

The User Profile

When you first run an Message Handling (MH) command, Message Handling (MH) creates the file **.mh_profile** in your **$HOME** directory. You can create personal Message Handling (MH) default values by modifying this file. The following list shows some of the Message Handling (MH) values that can be set in **$HOME**/.mh_profile:

Profile Entry	Description of the Entry
Path:	Sets the user directory for Message Handling (MH) transactions. This entry must appear in **$HOME/.mh_profile**. When a Message Handling (MH) command creates **$HOME/.mh_profile**, the Message Handling (MH) system automatically creates a **Path:** entry with the default value **Mail**. Throughout the Message Handling (MH) documentation, *user_mh_directory* refers to the value of the **Path:** entry.
context:	Sets the name of the Message Handling (MH) context file. Unless a full path name is specified, this file resides in *user_mh_directory*. The default file name is *user_mh_directory*/**context**.
Editor:	Specifies the initial editor to be used by the **comp**, **dist**, **forw**, and **repl** commands. The default editor is **prompter**.
Msg-Protect:	Sets the protection level for message files. The protection level must be specified in octal form. The default is `644`.
Folder-Protect:	Sets the protection level for folder directories. The protection level must be specified in octal form. The default is `711`.
cmd:	Specifies the flags to be used whenever the Message Handling (MH) command *cmd* is run. You can use this entry to override **.mh_profile** settings for particular commands. For example, the following entry specifies that the **comp** command is to invoke the `e` (INed editor) command regardless of the editor specified in the **Editor:** entry: `comp: -editor e`

lasteditor-**next:** Specifies the editor to be used for re-editing a draft after initial editing with *lasteditor*. When *lasteditor* is used as the initial editor by the **comp**, **dist**, **forw**, and **repl** commands, the default editor for re-editing the draft is the editor specified in this profile entry. This editor takes effect at the **whatnow** level of the **comp**, **dist**, **forw**, and **repl** commands. For example, the following profile entry states that the `e` (**INed** editor) command is the default re-editing command to be invoked for drafts initially edited using `prompter`.

```
prompter-next: e
```

Alternate-Mailboxes: Specifies the addresses that belong to you. The **repl** command uses this profile entry to determine which addresses to include in replies. The **scan** command uses this entry to determine which messages originated from you. When specifying addresses in this entry, you must separate each address with a comma and list the official host names for the addresses (local nicknames are not resolved to their official host names). If you do not specify the host name for an address, **repl** and **scan** regard that address on all host systems as belonging to you. The * (asterisk) can be used at either or both ends of the host name and the mailbox to indicate wild-card matching. The default is your user ID.

Draft-Folder: Specifies the default draft folder for the **comp**, **dist**, **forw**, and **repl** commands.

MailDrop: Specifies your maildrop for use with the **inc** command. The **$MAILDROP** environment variable supersedes this profile entry. The default is **/usr/mail/$USER**.

Signature: Specifies your mail signature for use with the **send** command. The **$SIGNATURE** environment variable supersedes this profile entry.

See the **mh-profile** section in *AIX Operating System Technical Reference* for more information about the **.mh_profile** and a more complete listing of the profile entries that you can define.

Format Files

The MH package enables you to maintain multiple formats for different types of messages. You can maintain formats for new messages, for redistributed messages, for replies to messages, for forwarded messages, and for message digests. You can create and modify system formats and personal formats. The section **mh-format** in *AIX Operating System Technical Reference* describes the Message Handling (MH) format strings that you can place in a format file.

Using Special Features of the MH Package

The MH package provides you with several special features. The following sections discuss two of these features: your ability to specify messages in many ways and your ability to maintain drafts of messages.

Specifying Messages

Many Message Handling (MH) commands allow you refer to particular messages. You can specify messages in the following ways:

- By stating the numerical name of the message. When you store a message in a folder, the MH package provides the message with a numerical file name. You can use this numerical name when referring to that message.
- By stating a Message Handling (MH) keyword that indicates a message. You can use the following keywords to specify messages:

Keyword	Description
first	The first message in the folder. The lowest message number is **1**, but a folder does not have to contain a message by this number.
prev	The message that is numerically previous to the current message.
cur	The current message.
.	The current message.
next	The message that is numerically next after the current message.
last	The last message in the folder.
new	A new message. This is equivalent to the message following the **last** message in the folder. You can not use this keyword when specifying a range of messages
all	All of the messages in the folder.

  For each of these keywords, if the folder is empty, the keyword represents message **1**.
- By stating the sequence name that represents one or more messages. Each sequence is associated with a particular folder and can be used to refer to messages in that folder only. You can specify message sequences that are designated as public sequences, message sequences that you define, and message sequences that the MH package system keeps track of for you.
- By describing a range of messages.

Defining a Message Sequence

You can use the **pick** and **mark** commands to define a message sequence. The name you choose for a sequence can contain alphabetic characters only and can not conflict with one of the Message Handling (MH) keywords. For example, you can define a sequence named `frombrenda`, but you can not define a sequence named `new`. You can define a maximum of about 10 sequences for each folder.

See the **pick** and **mark** sections of the *AIX Operating System Commands Reference* for more information about defining message sequences.

Using Sequences Defined by the MH Package

The MH package optionally keeps track of the messages that you previously specified and the messages that you have incorporated into a folder, but that you apparently have not seen. You can specify that you want the MH package to keep track of these messages by defining the entries **Previous-Sequence:** and **Unseen-Sequence:** in your **.mh_profile**.

For example, you can place the following definitions in **$HOME/.mh_profile**:

```
Previous-Sequence: pseq
Unseen-Sequence: unseen
```

These definitions enables you to refer to the sequences `pseq` and `unseen` for each of your folders. The sequence `pseq` refers to the last message or sequence of messages that you specified. The sequence `unseen` refers to the messages for which you have run the **inc** command, but for which you have not run the **show** command.

Specifying a Range of Messages

You can specify a range of messages in the following ways:

- By specifying the explicit range of the messages in the sequence. You can specify an explicit range by placing a `-` (dash) between the message identifier beginning the range and the message identifier ending the range. For example, the following command requests a scan listing of messages `9` through **last** inclusive:

  ```
  scan 9-last
  ```

  When you specify an explicit range, the range must contain messages.

- By specifying the general range of messages in the sequence. You can specify a general range by placing a `:` (colon) between the message identifier beginning or ending the range and the number of messages in the range. If the message identifier is **prev** or **last**, the message identifier ends the range. Otherwise, the message identifier begins the range. For example, the following command requests a scan listing of the last five messages:

  ```
  scan last:5
  ```

The following example requests a scan listing of the first three messages:

```
scan first:3
```

You can use the `+` (plus) and `-` (minus) symbols to change (or explicitly state) the direction of the range. The `+` directly following the `:` indicates that the range begins with the message identifier. For example, the following command requests a scan listing of the message **prev** and the three messages following **prev**:

```
scan prev:+4
```

The `-` directly following the `:` indicates that the range ends with the message identifier. For example, the following command requests a scan listing for the five messages preceding message `23` and for message `23`:

```
scan 23:-6
```

You can not use the keyword **new** when specifying a range. If you specify a message that is greater than the **last** message in the folder, the MH package interprets that message as the message number following **last**.

Creating and Using Message Drafts

When you create a message using the commands **comp**, **dist**, **forw**, and **repl**, the MH package enables you to save the message in a folder until you are ready to send the message. You can specify a name for this folder using the **-draftfolder** flag. Until you send the message, the message is considered a ***message draft***. When you send the message, the MH package renames the message draft by placing a comma before its file name. This renaming of the message removes the message from active draft status. (The profile entry **rmmproc:** is not consulted for this automatic renaming.)

A message draft is similar to other Message Handling (MH) messages. You can store several message drafts in a folder, you can create sequences containing message drafts, and you can run Message Handling (MH) commands (such as **refile**, **rmm**, and **scan**) with a message draft as an argument.

The **-draftfolder** flag does not change the current folder or the current message.

Chapter 11. Managing Distributed Services

CONTENTS

About This Chapter

This chapter describes the Distributed Services program and how to use it in some specific applications. It includes information about:

- What Distributed Services is
- What you need to install it
- How to configure a basic installation
- How to configure and use an installation that appears to look like one system (single-system image)
- How to configure and use a print and batch server system
- How to configure and use a central system to provide program code to others on the network (code server)
- How to use Distributed Services configuration commands.

Appendix B, "Distributed Services Customization Forms" on page B-1 provides forms to help you plan for installing a distributed system.

Note: In the examples in this chapter, all user-written shells are assumed to:

- Belong to UID 0 and GID 0
- Reside in the **/etc** directory
- Have the permissions **754** (read, write, execute by owner; read, execute by group; read by others).

Information Requirements

Before reading about Distributed Services, you should understand the principles of AIX system management described in the following topics:

- "The File System—Background for System Management" on page 1-6 and "The Base AIX File System" on page 1-12
- "Managing User Accounts" on page 2-13
- "Information about File Systems—The /etc/filesystems File" on page 2-36
- "Backing up Files and File Systems" on page 2-45

For information on adding an RT to a communications network, see the Communications Planning topic in the *Planning Guide.*

Prerequisite Components

Before configuring Distributed Services to run in your system, ensure that you have the following hardware and software installed and operating on your system.

Install one of the following adapter cards connected to other RT systems on a network:

- Baseband Adapter
- IBM Token-Ring Adapter
- Multiprotocol Adapter.

If there are no conflicts with other devices, configure the Baseband Adapter to match the default values that **devices** uses:

- All address jumpers disconnected
- Interrupt level set to 3.

If you have a Personal Computer AT Coprocessor, you may need to use a different bus address than the default. If you have a 3278/79 Emulation Adapter, you may need to resolve interrupt level conflicts between it and the Baseband Adapter. You may also need to consider other DMA or interrupt level conflicts as described in *Options Installation.*

Install and configure the following software:

- AIX Operating System licensed program (see *Installing and Customizing the AIX Operating System*)
  - VRM Program
  - Base System Program
  - VRM Device Drivers (at least one of the following):
    - VRM Baseband Adapter Device Driver
    - Token Ring Adapter Device Driver
    - Multiprotocol Adapter Device Driver
- SNA Services.
- Distributed Services licensed program

In addition, customize your AIX Operating System to include the following:

- Run the **devices** command to add the adapter and data link type that matches your network connection (see Figure 11-1 on page 11-7). See "Customizing System Devices" in *Installing and Customizing the AIX Operating System.*
- Optionally, you may want to install TCP/IP. You can use this program to test network operation and, by using it to log in to the remote system, help make configuration easier. See *Interface Program for use with TCP/IP* for more information about this program.

- Use **updatep** to apply any updates to the software that you have installed.
- Run the **minidisks** command to add local file systems (see "Customizing System Minidisks" in *Installing and Customizing the AIX Operating System*).
- Run the **users** command to add local users (see "Managing User Accounts" on page 2-13). If installing a single-system image configuration, do not add users at this time.

Adapter Number	Device Name	Data Link Name	Attachment Profile	Physical Link Profile
Baseband Adapter (Ethernet):				
First	net0	ethllc0	EDEFAULT	EDEFAULT
Second	net1	ethllc1	DDEFAULT	EDEFAULT1
802.3 (Ethernet):				
Port 1	net0	eth3llc0	HDEFAULT	HDEFAULT
Port 2	net1	eth3llc1	IDEFAULT	HDEFAULT1
IBM Token-Ring Adapter:				
First	token0	trllc0	KDEFAULT	TDEFAULT
Second	token1	trllc1	LDEFAULT	TDEFAULT1
Multiprotocol Adapter (SDLC):				
Port 1	mdp0	sdlcllc0	SDEFAULT	DDEFAULT
Port 2	mdp1	sdlcllc1	ODEFAULT	DDEFAULT1
Port 3	mdp2	sdlcllc2	PDEFAULT	DDEFAULT2
Port 4	mdp3	sdlcllc3	QDEFAULT	DDEFAULT3

Figure 11-1. Data Link Types

Understanding Distributed Services

Using the mount capabilities of AIX, you can create file trees that are independent of the file systems on individual minidisks. In addition, the Distributed Services licensed program allows you to use both local and remote directories and files to build these file trees. The following terms apply to AIX directory and file structures possible with the enhanced AIX mount capabilities.:

file system
: The complete structure of directories and files contained on a single minidisk. To mount a file system, you mount the device that contains it (for example: `mount /dev/hd7 /mnt`).

virtual file system
: The structure created by mounting a directory or file (as opposed to mounting a minidisk). Each directory or file mount creates a new virtual file system.

file tree
: The complete directory and file structure of a particular node, starting at the root directory (/). A file tree is the product of all minidisk, directory, and file mounts that have been performed. The path name to any accessible directory or file is determined by the structure of the file tree.

These additional terms apply to a Distributed Services network:

node An individual system connected to the network.

client In an interaction between nodes, the node that requests resources.

server In an interaction between nodes, the node that provides resources.

As an example of what you can do with Distributed Services, the following three figures represent the creation of a file tree (on node D174) from parts of different file systems. Each file system resides on a different node in a Distributed Services network. First, each node mounts its file system that contains the **/u** directory. Figure 11-2 on page 11-9 shows a portion of this file system for each node.

Next, node D174 mounts a directory from node D145 with the command:

```
mount -n D145 /u/tom/proj_data /u/joe/data
```

This mount creates a virtual file system (`/u/joe/data`). The file tree on D174 now includes a remote directory structure. Generally, it does not matter to a user whether a particular directory or file is local or remote. Figure 11-3 on page 11-10 shows the resulting file tree.

Finally, node D174 mounts a directory from node 211 with the command:

```
mount -n D211 /u/sam/proj3_data /u/joe/data/proj3
```

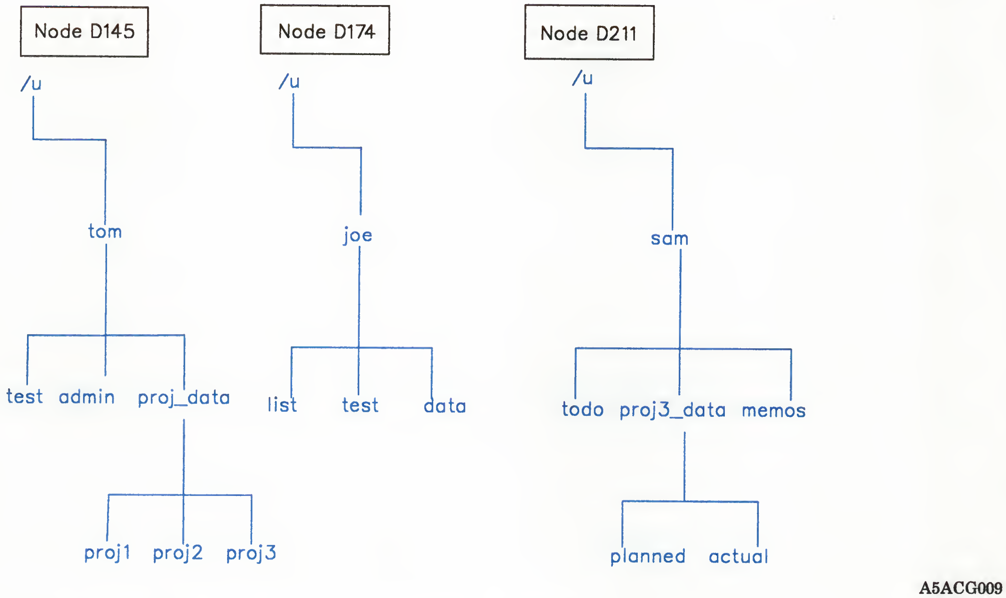


Figure 11-2. Creating a File Tree: /u File System Mounts

Figure 11-4 on page 11-11 shows the resulting file trees from that mount. The structure below the `/u/joe` directory now contains directories and files that physically reside on three different nodes. However, the file tree created by the directory mounts works just as it would if the same structure existed on a single (minidisk) file system.

Depending upon your requirements, your Distributed Services network can be simple or elaborate. If, for example, your network is small and its users perform their own mounts from the command line, your Distributed Services network will be relatively simple to configure and maintain. However, if you have a larger network and require a high degree of uniformity from node to node, your planning, configuration, and maintenance of Distributed Services will be more complex.

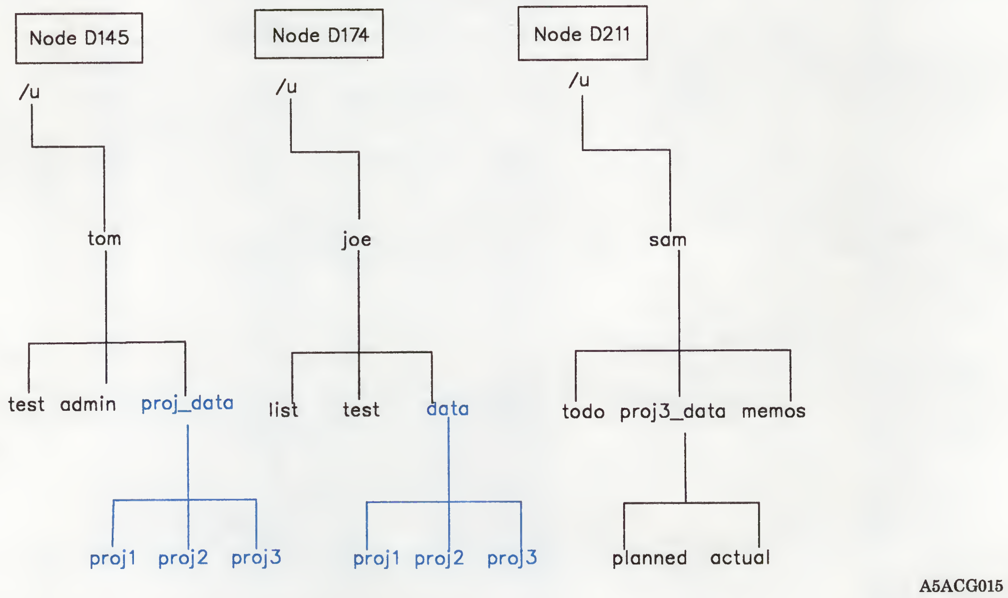


Figure 11-3. Creating a File Tree: Directory Mount

Distributed Services File Systems

Note: Minidisks must be mounted locally before any of their contents can be mounted by remote nodes.

The **/etc/filesystems** file can contain stanzas for file and directory mounts just as it does for file system mounts. Also, a user with the appropriate authority can perform directory and file mounts from the command line.

Each **mount** command creates a description of a virtual file system and adds the description to a master list of virtual file systems on that node. Each **umount** command removes a description from the list. The description contains information about the mounted object, the object that is mounted over, and the relationship of the virtual file system to other virtual file systems in the list.

A ***vnode*** describes a file or directory in a specific file system (since the same file or directory can be part of more than one file system). The vnode points to the i-node that describes the file on the minidisk that contains it.

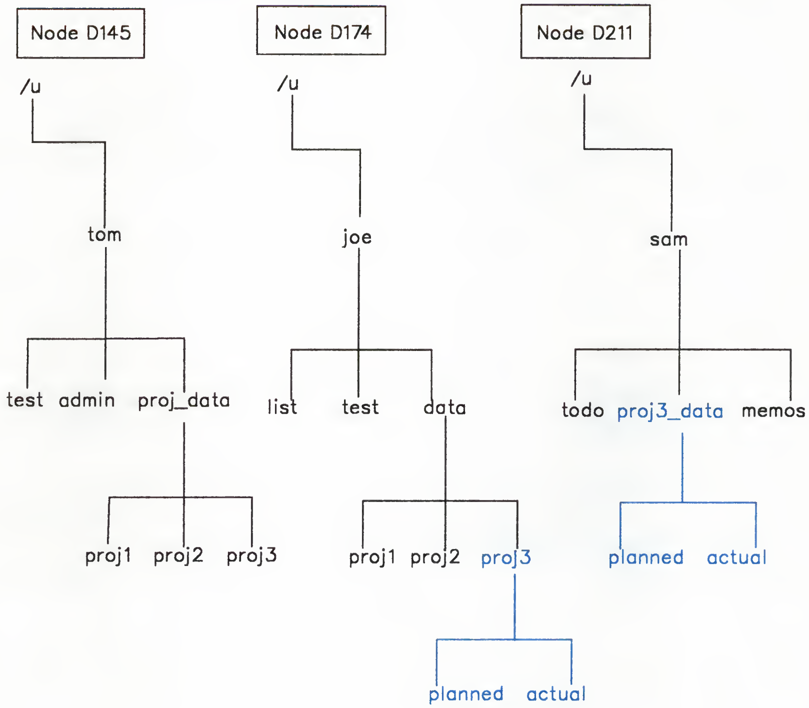


Figure 11-4. Creating a File Tree: The Completed Structure

The file system ***subtree*** is the structure of vnodes beneath each mounted directory. Each vnode contains a pointer to the virtual file system description. The file system of a particular system consists of all subtrees created by mounts on that system.

A ***file handle*** contains the file's device number, i-node number, i-node generation number, and type. The file handle, combined with the node ID of the system that contains the file, uniquely identifies the file on a specific system.

The following system maintenance commands can be run only on file systems (that is, on minidisks, not on virtual file systems or file trees):

- **backup -m**
- **restore -m**
- **fsck**

Mounting and Unmounting Files and Directories

When used to mount directories or files, the general format of the **mount** command is:

`mount` *pathname1 pathname2*

where *pathname1* is the path name of the object to be mounted and *pathname2* is the path name of the object to be mounted over.

The following flags to the **mount** command apply specifically to file and directory mounts:

-n *nodename* Specifies a remote system. If the object to be mounted (*pathname1*) is remote, you must use the **-n** flag in a command of the form:

`mount -n` *nodename pathname1 pathname2*

nodename is the name of the remote system that contains the object to be mounted. *pathname1* is the path name to the object on the remote system. *pathname2* is the path name to the object on the local system to be mounted over. If run without parameters, **mount -n** *nodename* displays the mount table at the remote node, *nodename* (the remote equivalent of running **mount** without flags locally).

-t *string* Specifies mounts of **type** *string*. **type** is an attribute that can be included in the **/etc/filesystems** stanza.

-i Causes a remote file tree structure to be ***inherited***. That is, the **-i** flag enables **mount** to replicate, on a local node, the structure of a remote file tree. The inherited structure includes all file and directory mounts below the mounted directory (*pathname1*). To replicate the structure, **mount -i** individually performs each directory or file mount contained in the file tree to be inherited.

The **umount** command unmounts files and directories as well as local minidisks. To unmount a file or directory, use a **umount** command of the form:

`umount` *pathname2*

where *pathname2* is the the file or directory over which another file or directory is mounted).

Note: You cannot unmount a file or directory by naming the path name of the mount point (for example, *pathname1* in the previous **mount** command example). You can, however, unmount a device by naming the path name of the mount point (for example, `umount /u`).

To unmount only the remote mounts from a specific node, use the **-n** flag, for example:

`umount -n` *nodename*

To unmount all remote mounts, use the **allr** flag, for example:

```
umount allr
```

The **umount all** command unmounts all remote files and directories as well as all local minidisks. The **umount -t** *type* command unmounts all mounted objects listed in **/etc/filesystems** that have a `type` attribute of *type*. The **umount -f** command forces unmounting of all mounts below the directory named. See *AIX Operating System Commands Reference* for more information about the **umount** command.

Users can issue arbitrary file or directory mounts (**mount** *directory1 directory2*, **mount** *file1 file2*) or mounts described in **/etc/filesystems** if they have the following:

- Read and write permission for the object being mounted over.
- Read or write permission for the object being mounted.

A member of the system group can also issue any mount described in the **/etc/filesystems** file. A user with superuser authority can issue any valid **mount** command.

Users can unmount any mount that they are allowed to issue. Members of the system group can unmount any mount.

The remainder of this section discusses ways to use the file system mount capabilities in a Distributed Services environment.

Mounting File Trees That Contain Mounts

Once a minidisk file systems are mounted, you can assemble objects from them into file trees with directory and file mounts. Ordinarily, when you mount a directory, you mount only some part of a file system, not a file tree. However, you can replicate file trees by using ***inherited mounts***. For example, if a directory has two directories mounted in its subtree, then:

- An ordinary **mount** command performs only one mount, that of named directory.
- An inherited mount performs three mounts: the named directory and then each of the two directories mounted in its subtree.

Thus, a single inherited mount command may result in numerous individual mounts.

There are two ways to perform inherited mounts:

- Issue the **mount -i** command.
- Place the **mount = inherit** attribute in the appropriate **/etc/filesystems** stanzas.

Inherited mounts are convenient when:

- Users at different nodes require access to the same data, and
- The physical location of the data may have to be changed regularly to balance system requirements and resources.

With inherited mounts, you can rearrange the location of data in one file system without requiring users at other nodes to update their **/etc/filesystems** file or automatic mount procedures.

The other nodes have a single **/etc/filesystems** stanza to describe the mount. However, inherited mounts are performed only when the mount command runs; that is, if the file tree is modified while the system is in use, the inherited mounts are not dynamically updated. Therefore, when it is necessary to change file trees that are inherited by other nodes, it is best to inform users that the changes will be made at specific times.

An inherited mount consists of multiple, independent directory or file mounts. Permissions remain the same for inherited mounts. They apply only to the root of the mounted file tree. You do not need read or write permission to all the directories or files in an inherited mount to perform one.

To unmount an inherited mount, you must do one of the following:

- Unmount each directory or file individually, in the reverse order from that in which it was mounted.
- Use the **umount all** command.

When a node becomes inactive and that node is part of the inherited mount sequence, you may not be able to unmount files or directories in that sequence with the normal forms of the **umount** command. Use the **-f** flag to force an unmount and clear up the local mount table:

`umount -f -n` *nodename file*

You cannot unmount an inherited mount by naming the directory that was originally mounted, or by using the **umount allr** or **umount -n** commands. Generally, it is most convenient to design inherited mounts that:

- Are performed automatically at system start
- Do not require modification while users are logged into the system
- Can be unmounted with the **umount all** command as part of shutting down the system.

Performing Routine Mounts Automatically

The way in which remote mounts are managed depends upon the nature of the particular Distributed Services network. In some cases, it may be sufficient simply to allow users to perform mounts as needed; that is, there is no set network file system structure that must be in place in order for the network to be useful. In other cases, however, it is convenient to create a file system structure that automatically incorporates routine remote mounts. This section discusses techniques that can help you manage remote mounts.

If you routinely mount specific remote directories, it is convenient to mount them automatically. Because the nodes in a Distributed Services network may start at different times, relative to each other, remote mounts may fail when they are first issued. This

section explains how to develop an automatic mount procedure that continues to issue the mounts until they succeed.

An automatic mount procedure involves the following AIX and Distributed Services components:

/etc/filesystems The file containing file system descriptions (see "Information about File Systems—The /etc/filesystems File" on page 2-36).

type = *string* An attribute that can appear in an **/etc/filesystems** stanza.

-t *string* A **mount** argument that causes the **mount** command to mount all directories or files whose **filesystems** stanzas contain the attribute **type** = *string*).

In addition, when you specify the **-t** flag with a file or directory mount, the **mount** command checks to see if that file or directory is already mounted. If it is, **mount** does not attempt to mount it again.

This flag is necessary because the **mount** command treats duplicate mount requests differently depending on whether it is a file system mount or a file or directory mount. If a file system is already mounted, **mount** will not mount it again. However, **mount** does duplicate file or directory mounts unless you specify the **-t** *string* argument.

shell programs User-written programs that issue remote mounts, monitor their status, and re-issue or reschedule remote mounts, as needed.

/etc/rc.ds An initialization file run by **init** to configure Distributed Services at system start. (This is also a shell program.)

For a sample automatic mount shell program, see "Automatic Mount Procedures" on page 11-89.

Looking at Covered Files and Directories

When you create a remote mount, it covers up any files or directories on the local system that are at or below the mount point. At times you may want to look at the files that are covered up without unmounting the remote file system.

If Distributed Services is installed on your system, when the system starts it mounts the root directory onto the **/native** directory before it does any of the remote mounts. Thhis path to the root directory is not affected by later remote mounts over parts of the actual root directory. You can access any of the local files through the **/native** path. For example, if there were a remote mount over the **/usr/man** directory, you can view the original directory by looking at **/native/usr/man**.

Starting Distributed Services

When the system starts, it normally executes instructions in **/etc/rc** and **/etc/rc.include** to set initial conditions for the system. If the file **/etc/rc.ds** exists, the system also executes instructions in that file. The instructions in **/etc/rc.ds** perform the following functions:

- Start SNA
- Start a default attachment for the network adapter
- Start Distributed Services.

This file is installed with Distributed Services. It comes configured to start an attachment for Baseband Adapter. If that default attachment does not match your network, change **/etc/rc.ds** to match your network using the following procedure. After changing this file, shut down the system and start it again to enable Distributed Services. After Distributed Services starts, you must configure it for the systems in your network before you can use it.

Changing /etc/rc.ds

1. Edit **/etc/rc.ds** with your favorite editor.
2. Find the line that starts the `EDEFAULT` attachment:

   ```
   start /attachment/EDEFAULT &
   ```

3. Insert a comment character (#) in the first character of that line to prevent starting the attachment for Baseband Adapter:

   ```
   #  start /attachment/EDEFAULT &
   ```

4. Find the line that starts the attachment for your network attachment (see Figure 11-1 on page 11-7). For example, for the first token-ring attachment, find:

   ```
   #  start /attachment/KDEFAULT &
   ```

5. Remove the comment character (#) from the first character position of that line to enable starting the attachment:

   ```
   start /attachment/KDEFAULT &
   ```

6. Save the file and exit the editor.

Configuring a Basic Distributed Services System

Once you have installed the hardware and software needed to operate Distributed Services, you can configure a basic distributed system using the procedure described below. This procedure creates a working system that has the following restrictions:

- Access permissions are limited. See "Changing Network Users/Groups Table Entries" on page 11-110 to change access permissions.
- No communications password is used. See "Working with the Distributed Network Node Table" on page 11-96 to add a password to the connection.

Once the basic system is configured and working, you can mount files and directories from the other systems in your network using manually entered **mount** commands. Other procedures in this chapter explain how to change this basic system to:

- Mount remote files and directories automatically (see "Setting Up Automatic Remote Mounts" on page 11-81).
- Enforce access permissions more carefully (see "Working with the Network Users/Groups Table" on page 11-107)
- Set up a password-protected session with another system (see "Working with the Distributed Network Node Table" on page 11-96).

Note: The Distributed Services configuration commands use a full screen presentation similar to that used for Usability Services. The command keys are the same as those used for Usability Services. Refer to the keyboard template for Usability Services for key assignments. You can also use the mouse to select fields on the screen.

Configuring a Basic System

The following procedure assumes that Distributed Services is running (see "Starting Distributed Services" on page 11-16).

1. Create a nickname for each system in the network and write them down.
2. Use the **uname -m** command on each system in the network to determine their node IDs. Write the node IDs next to the system nicknames.
3. Use the **ndtable** command to define each node in the network.
4. On each system, use the **ugtable** command to authorize users and groups from other systems to have access to the local system.
5. Use the **mount** command to test the configuration with a trial remote mount.

More Detailed Information

1. **Create Nicknames**

   You can name systems in the network with any name that you choose. The name must be made up of only the letters of the alphabet and numbers. If you are using TCP/IP, choose only lowercase letters for the nickname. Do not choose a nickname that is the same as the user name of a user on the system. For consistency, use the same nickname for all of the following uses:

   - Distributed Services nickname
   - Node name - This is the name returned by the **chparm nodename** command. Set this name with the **chparm** command. For example, to set the node name to `george`, use the command:

     ```
     # chparm nodename=george
     ```

     The name change takes effect the next time that you start the system. If you change the node name, also change the `node` name in **/etc/master** to the same value.
   - TCP/IP host name - This name is returned by the **hostname** command if you have TCP/IP installed. Set this name by editing **/etc/rc.tcpip** and changing the line that looks like the following line so that `sample` is the correct name and `sub2.sub1.dom` indicates the correct domain address:

     ```
     /bin/hostname sample.sub2.sub1.dom      >/dev/console
     ```

   Once you have decided on a nickname for each system, write the nicknames down in a vertical list. For example:

   ```
   george
   kronos
   athena
   hera
   zeus
   ```

2. **Determine Node IDs**

   The node ID is a unique number that is stored in ROM (Read Only Memory) on the processor board of your system. It identifies your system among all other RTs. This ID changes only if you change the processor board in your system. Use the **uname -m** command to determine this number. For example, the following sequence occurs on system `athena` to determine the node ID:

   ```
   # uname -m
   108133EB
   ```

The number returned (108133EB) is the node ID for that system. Repeat this command on all systems in your network and write the number next to the corresponding nickname on your list:

```
george          2081043B
kronos          108133F5
athena          108133EB
hera            208131AA
zeus            2081314A
```

3. **Define Each Node**

   Use the **ndtable** command to add each of the systems in the network to the Distributed Services node table at each system. See "Working with the Distributed Network Node Table" on page 11-96 for detailed information about adding a node. For the example network, perform the following steps on each system in the network:

   a. Enter the **ndtable** command. A pop-up panel appears, requesting the nickname of the node to configure (the local node is the default).

   b. Press **Do**. Another pop-up panel appears, requesting whether you want a static or a dynamic table (static is the default).

   c. Press **Do**. The node table appears. If you have not defined any nodes, it contains only an entry for the local node.

   d. On the command bar, select **Add**. A pop-up panel appears for entering the information for one node.

   e. Type in the nickname for one node other than the local node.

   f. Press **Tab** to move to the next field.

   g. Type in the node ID that corresponds to the nickname entered.

   h. Press **Tab** enough times to move to the field for selecting the type of Data Link that your network uses. Select that data link type by moving the cursor to the data link type and pressing **Select**. Another pop-up panel appears.

   i. Press **Do** to select the default values from the pop-up panel.

   j. Press **Do** to complete adding the node to the table. The system displays the node table with the information for the added node entered in the table.

   k. Repeat steps 3d through 3j for each of the remaining nodes in the network (except the local node).

   l. On the command bar, select **Close** to return to the operating system.

4. **Define Users and Groups**

   Use **ugtable** to authorize user IDs and group IDs to operate between systems. For a minimum system, authorize the **guest** user ID and the **usr** group ID. You may also

want to authorize other IDs, like your own user ID and the group IDs of the groups to which you belong. If you are not concerned with security, you may also authorize user ID 0 (root) and group IDs 0 and 1 (system and staff). If you want to install the single-system image configuration (see "Creating a Single-System Image" on page 11-29), just authorize **guest** and **usr**. Use the following steps at each system on the network to authorize IDs:

a. Enter the **ugtable** command. A pop-up panel appears, requesting the node to configure (the local node is the default).

b. Press **Do**. The user-group table appears. If you have not defined any IDs, it is empty.

c. On the command bar, select **Add**. A pop-up panel appears for entering the information for one ID. Use the **Tab** key to skip over fields that are not mentioned in the following steps.

d. Select the `User` button.

e. Enter `guest` in the `User/Group Name` field.

f. Enter `*` in the `Inbound Network ID` field.

g. Enter `*` in the `Originating Node ID/Nickname` field.

h. Press **Do** to add the `guest` user ID. This addition means that any request from another system that is not specifically authorized to have access will have the privileges of **guest** user ID. The user-group table appears with the new information added.

i. On the command bar, select **Add**. A pop-up panel appears for entering the information for one ID.

j. Select the `Group` button.

k. Enter `usr` in the `User/Group Name` field.

l. Enter `*` in the `Inbound Network ID` field.

m. Enter `*` in the `Originating Node ID/Nickname` field.

n. Press **Do** to add the `usr` group ID. This addition means that any request from another system that is not specifically authorized to have access will have the privileges of **usr** group ID. The user-group table appears with the new information added.

o. See "Working with the Network Users/Groups Table" on page 11-107 for information to add other IDs to the table.

p. Select **Close** to exit the table and return to the operating system.

5. **Test the Configuration**

   Use the **mount -n** command to test the new configuration. This command has the following format:

   `mount -n` *remote*

   In this format *remote* is the nickname that you assigned to one of the other systems on your network. When it executes, it displays the mount table from system *remote* on your local screen. For example, if you are at system athena, then the following command displays the filesystems that are mounted at system zeus:

   ```
   mount -n zeus
   ```

   You should be able to view the mount tables for all defined systems on the network from all defined systems on the network.

Customizing a Server Node

Take the following steps to customize each RT that allows remote access to local files or programs:

Planning a Server Node

1. Identify network requirements.
2. List the node ID (NID) of each RT that has remote access to local files and programs.
3. List the names of all network users and groups that have remote access to local files and programs. Include both the user or group ID and the node ID of the originating system.
4. Decide how to manage the translation of incoming and outgoing user and group IDs.
5. Build the necessary node table and ID table.
6. Check the permissions of local files and directories to insure that they provide or restrict network access as you intend.
7. Give client nodes the information they need to access local files and programs.

More Detailed Information

1. Identify network requirements. Consider the following:
   - To which files and directories do network users need access?
   - What kind of access is needed: search or execute permission, read permission, or write permission?
   - Which users need access: all users (local and remote), all network users, selected network and/or local users?
   - Which groups need access: all groups (local and remote), all network groups, selected network and/or local groups?
   - Who should be denied access?

2. List the node ID (NID) of each RT with remote access to local files and programs. To learn the NIDs, run the following command at each RT on the network:

   ```
   uname -m
   ```

   If you do not have access to all the RTs on the network, you will need to request this information from the local system administrator. If your network is part of a Baseband (Ethernet) LAN and you have installed TCP/IP, you may be able to use remote login to collect this information.

   As a general rule, you should also assign a 1- to 14-character nickname to each node. Associating a nickname with a NID can make it easier to build the tables or use the commands (such as **mount**) that require you to specify a NID.

   A node has only one valid NID, but it can have as many nicknames as it has client systems. Each system that accesses a node can assign it a different nickname. For example, Node A has NID `20810CBF`. Node B assigns the nickname `Tom` to Node A, while Node C assigns the nickname `Server`. The profiles at each system translate the locally defined nickname to NID `20810CBF`, which uniquely identifies Node A on the network.

3. List the names of all network users and groups with remote access to local files and programs. This list should include both the user or group ID and the node ID (or nickname) of the originating system.

4. Decide how you want to manage the AIX Operating System's translation of incoming and outgoing user and group IDs (see "Managing ID Translation" on page 11-118).

5. Build the necessary tables. Run the **ndtable** command to add a node profile for each node on the network to the Distributed Network Node Table. Run the **ugtable** or **dsldxprof** command to add translation entries to the Network Users/Groups Table.

   Note: If creating additional local user or group IDs is part of your plan to manage network IDs, add them to the **/etc/passwd** or **/etc/group** file before building the Network Users/Groups Table. (See the **users** command in *AIX Operating System Commands Reference.*)

   While you can create either table first, if you create the Network Users/Groups Table before you create the Distributed Network Node Table, you get a message with each entry warning you that the node is not known.

6. Check the permissions settings of local files and directories to insure that they provide or restrict network access as you intend.

   If you do not map network user or group ID 0 to local ID 0 (and as a general rule you should not), be sure the permissions of directories that are available for remote access are set to `search by others` (allowing other nodes to mount the directory) and to `read by others` (allowing users at other nodes to list the contents of the directory with commands like **li** or **ls**).

   Conversely, you can set the search permissions on a directory so that a remote user cannot mount or read that directory, but keep in mind that this will restrict local access as well. You can, however, use concurrent group membership at the local node and careful mapping of remote IDs to deny access to remote users while allowing it to local users, if such security is important.

7. Give client nodes the information they need to access your files and programs. Include the full path names of available resources, the path names that should be mounted as inherited mounts, and a list of network IDs.

Customizing a Client Node

Take the following steps to customize each RT that accesses remote files or programs:

Planning a Client Node

1. Identify network requirements.
2. List the node IDs of each RT that has files or programs that you plan to use.
3. If local users or groups have been assigned network IDs, list each user or group name and its corresponding network ID.
4. Decide where you want to mount remote directories or files and create empty directories or files to mount over, if necessary.
5. Build the necessary node table and ID table.
6. Add stanzas to the **/etc/filesystems** file to describe remote mounts.
7. Modify the **/etc/rc.ds** file to enable automatic or graceful startup, if desired, and to block all incoming requests (`dsstate -sb`), if you do not want other nodes to have access to your file systems.

Creating a Node Table Server

As a distributed network grows larger, maintaining separate copies of the Distributed Network Node Table at each node becomes a burdensome task. To help simplify the task of updating the Distributed Network Node Table and help ensure that each node has up-to-date information, set up one node as a ***node table server***. This node contains the master Distributed Network Node Table. All other nodes on the network simply mount the Distributed Network Node Table from the server node. Any changes made to the master Distributed Network Node Table are available to all nodes on the system without undue delay. The following paragraphs describe the procedures for setting up the server node and a client node.

Creating the Master Distributed Network Node Table

The master Distributed Network Node Table is the central data base for node connectivity information. It provides all information, except BIND passwords, to allow any system on the network to connect to any other system. Each system still maintains its own Network Users/Groups Table, and therefore, controls access to its files. Each system also maintains its own set of BIND passwords if the connections are set as `secure`. Use the following procedure to configure the Distributed Network Node Table of the server node.

Setting Up the Server Node

1. Use the **ndtable** command to define the master Distributed Network Node Table. This table must be a ***dynamic*** table. See "Working with the Distributed Network Node Table" on page 11-96 for information to define the network with this command. Also define any BIND passwords that the server system uses to connect to other systems.
2. Use the **ugtable** command to configure the server node to allow read access to the profiles data base for all systems on the network.

More Detailed Information

1. Use the **ndtable** command to define the master Distributed Network Node Table. When you start **ndtable**, it asks you for a Node ID Nickname. Use the default value (local node ID) and press **Do**. A second pop-up panel appears. Select **Dynamic** and press **Do** to enter the Distributed Network Node Table. Define the nodes in the network as described in "Working with the Distributed Network Node Table" on page 11-96. If any of the connections between the server node and another node are `secure`, you can enter the BIND password also. The password information is kept in a

separate profile that is not mounted as part of the node table server configuration. The password profile is not available to other nodes.

2. Use the **ugtable** command to configure the server node to allow read access to the profiles data base. All systems in the network must be able to mount and read the files in **/etc/profsvcs/pfslocal**. These files are normally owned by ID `root` and group `system`. The profiles that need to be mounted will be mounted during system start up. Set up a translation of the remote user ID of 0 to the local user ID of 0, and of the remote group ID of 0 to the local group ID of 0. Using a network ID of 0 for each of these translations, the entries in the Network Users/Groups Table might look like:

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
root	U	0	0	0	*
system	G	0	0	0	*

These entries give `root` and `system` access permissions for all files on the server system to anyone who has those permissions on any client system.

Setting Up a Client Distributed Network Node Table

The client Distributed Network Node Table is a mounted profile data base from the server node. To set up the client node to be able to mount the profile data base, perform the following procedure.

Setting Up the Client Node

1. Use the **ndtable** command to define the server node to the client system. This definition must be made in a ***static*** table. See "Working with the Distributed Network Node Table" on page 11-96 for information to define the server node with this command. Also define the BIND password (if the connection requires the additional security) used to connect to the server node only.
2. Use the **ugtable** command to configure the client node to be able to access the profiles data base on the server.
3. Use the **mount** command to mount the Distributed Network Node Table from the server onto the local Distributed Network Node Table.
4. Use the **pwtable** command to define BIND passwords (if needed) for connections between the client and the other (nonserver) systems on the network. See "Working with the Network Node Security Table" on page 11-104 for information to use this command.
5. Repeat steps 1 through 4 on the other systems in the network that are using the node table server.
6. Test the configuration by trying several remote mounts from each node.

More Detailed Information

1. Use the **ndtable** command to define the connection to the server node. When you start **ndtable**, it asks you for a Node ID Nickname. Use the default value (local node ID) and press **Do**. A second pop-up panel appears. Select **Static** and press **Do** to enter the Distributed Network Node Table. Define only the server node as described in "Adding a User or Group to the Network" on page 11-109. If the connection between the server node and the client node is `secure`, enter the BIND password also.
2. Use the **ugtable** command to configure the client node to allow read access to the master Distributed Network Node Table. Set up an outbound translation of the user ID of 0 to the user ID of 0 on the server, and of the group ID of 0 to the group ID of 0 on the server. Using a network ID of 0 for each of these translations, the entries in the Network Users/Groups Table might look like:

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
root	U	0	0	-	*
system	G	0	0	-	*

These entries give root and system access permissions for all files on the server system, but prevent anyone who has those permissions on any client system from accessing local files.

Note: You may need to allow inbound access for root for other configurations that you might also be using.

3. Use the **mount** command to mount the Distributed Network Node Table from the server onto the local Distributed Network Node Table. If the server is named zeus, the command that performs the mount looks like:

   ```
   # mount -n zeus -t ndtable
   ```

   To perform the mount automatically when the system starts, edit **/etc/rc.include** to remove the comment character (#) from the beginning of the following line:

   ```
   # mount -n Server -t ndtable
   ```

   Also, change Server in that line to be the nickname of the server node.

4. Use the **pwtable** command to define BIND passwords (if needed) for connections between the client and the other (nonserver) systems on the network. See "Working with the Network Node Security Table" on page 11-104 for information to use this command.

5. Repeat steps 1 on page 11-27 through 4 on the other systems in the network that are using the node table server.

6. Test the configuration by trying several remote mounts from each node. The following command displays the mount table for remote system *nodename*.

   # mount -n *nodename*

Creating a Single-System Image

Using Distributed Services, you can configure several systems to appear to be one system. Such a configuration is called ***single-system image***. When systems are configured in a single-system image, users of the system, users of external systems that communicate with the system and application programmers are *not* aware of differences between single- and multiple-system implementations. System administrators and maintenance personnel *are* aware of distinctions among systems.

Although some exceptions exist, Distributed Services implements a single-system image very closely. The systems are configured so that they share the files and directories that matter to the user and application programmer:

- **/etc/passwd** and similar files
- **$HOME** directories
- Directories containing application programs, commands and libraries (see also "Using Distributed Services to Provide Code Service" on page 11-54 for a method of sharing executable programs).

Configuring the systems with these guidelines accomplishes most of the properties of the single-system image. The login process at all systems is the same because **/etc/passwd** and related files are the same. Normal file system tasks and applications work in the shared directories. Administrative commands for ordinary users, such as **passwd**, also work properly.

Some administrative tasks must be done for each individual system. For example, you must install and configure AIX Operating System and Distributed Services on each system. Error logs are intentionally kept separately for each system; otherwise, the first problem determination step would be to find the faulty system. Some maintenance operations, such as image backups of disks and hardware diagnostics, must be performed on individual systems while that system is in maintenance mode.

However, you perform other tasks only once for all the single-system image systems. For example, the **users** command creates an entry in **/etc/passwd**, creates a home directory and copies standard files to the home directory. You only need to run the **users** command once to add a user to all systems in the single-system image configuration. In addition, you can do routine maintenance, such as backing up and restoring files, for the system as a whole while the system is in normal operation.

The procedures in this section help you install a single-system image with a network of RTs using Distributed Services. The resulting configuration has the following characteristics:

- The order in which the systems start up is not important. There is a delay, however, from the time that all systems are individually operating until the time that the single-system image configuration is complete.

- You manage the configuration by controlling a few configuration files. You can add or delete systems and users, and make other configuration changes by changing these configuration files.
- The configuration uses no additional entries in **/etc/filesystems** for single-system image purposes.
- All systems use the same configuration files. The configuration files on the administrative system (the system that provides **/etc/passwd** and related files) are the same as those on the other systems.
- To change the administrative system from one system to another requires only changing a line in one configuration file, waiting for a few minutes for the configuration file to be updated on all systems, and then shutting down the old administrative system.
- Client systems recognize when the server is not available, and switch to local copies of administrative files.
- Each system maintains copies of the administrative files and updates them periodically in case the administrative system goes down.
- If the system that provides your $HOME directory is not available, you find out when you log in. You can then use an alternate directory or wait until the system becomes available.
- Mail sent to you is delivered to you regardless of which system in the single-system image configuration that you use to log in.
- Named printers point to the same physical printer regardless of which system is being used. Printing without naming a printer uses the printer connected to the local system.

When you start the systems configured for single-system image, they run a new start up file, **/etc/rc.DS**. This file starts SNA Services, Distributed Services and runs the programs required to set up the single-system image. During start up, all messages are written to **/usr/adm/ds.msg** to provide a log of what has happened to help determine if something went wrong.

Defining a Single-System Image Configuration

Notes:

1. The following procedures assume that you have successfully installed Distributed Services on all systems involved in the configuration using the procedure described in "Configuring a Basic Distributed Services System" on page 11-17. You need only have the minimum user and group ID translations defined as described in Step 4 on page 11-19.
2. You must be operating with superuser authority to perform these procedures.

Creating a single-system image environment requires that you perform some configuration tasks at each system that is to be a part of the environment. The actual tasks depend upon whether the system is to be the ***administrative system*** or a client system. The administrative system is the server system for important control files, such as **/etc/passwd**, and for the mounted directories containing libraries and commands. You must choose one of the systems to be the administrative system and configure it before configuring any of the other (client) systems. "Configuring the Administrative System" on page 11-34 describes the procedure to configure an administrative system. "Configuring a Client System" on page 11-41 describes the procedure to configure a client system.

Example configuration files are contained in a file in **backup** format called **/usr/lpp/ds/samples/SSI.backup**. When you unpack this file while configuring the administrative system, it adds the files shown in Figure 11-5 to the file tree in **/tmp**.

File	Description
Information Files:	
SSI.README	This file contains information describing the single-system image configuration. It describes how to use a single-system image environment, what's contained in and how to use a backup.
Configuration Files:	
/etc/adminserver	This file contains the node name of the current administrative system (the system that provides **/etc/passwd** and related files).
/etc/SSImachines	This file contains a list of the node names of all system in the single-system image configuration, including the node name of the administrative system.

Figure 11-5 (Part 1 of 4). Single-System Image Files

File	Description
/etc/server.files	This file contains a list of files to mount from the administrative system (named in **/etc/adminserver**).
/etc/server.dirs	This file contains a list of directories to mount from the administrative system (named in **/etc/adminserver**). This file may be empty, but it must exist.
/etc/remounts.list	This file contains a list of files and directories to unmount when the administrative system cannot be accessed. These files and directories are mounted again when the administrative system becomes available.
/etc/ug.SSI	This file contains a list of user and group ID translations in a form that can be input to **dsldxprof** (see "Using the dsldxprof Command" on page 11-121). This file is used by **/etc/SSImounts** to update the Network Users/Groups Table when the single-system image configuration starts.
/etc/oug.SSI	This file is the old version of **/etc/ug.SSI**. This file is used by **/etc/SSImounts** to determine if the Network Users/Groups Table must be updated.
/usr/adm/user.cfile	This profile defines the default values that the **users** command uses when adding new users to the system.
Program Files:	
/etc/rc.DS	This shell script is run by **/etc/rc.include**. It ages the message logging files (**/usr/adm/ds.msg**) and runs **/etc/ssi** mounts.

Figure 11-5 (Part 2 of 4). Single-System Image Files

File	Description
/etc/SSImounts	This shell script is run by **/etc/rc.DS**. When it runs, it: 1. For systems other than the administrative system, it: a. Copies the content of files listed in **/etc/server.files** from the administrative system to the local system. b. Mounts the files listed in **/etc/server.files**. c. Mounts the directories listed in **/etc/server.dirs**. 2. Starts the **/etc/unmountd** daemon. 3. If **/etc/ug.SSI** is different from **/etc/oug.SSI**: a. Runs **dsldxprof** using **/etc/ug.SSI** as input to update the Network Users/Groups Table. b. Copies **/etc/ug.SSI** to **/etc/oug.SSI**. 4. Mounts $HOME directories that are stored on other systems.
/etc/umountd	This daemon tries to open files on the administrative system every 60 seconds. If it cannot open the files, it runs **/etc/remounts**. This daemon also updates local copies of mounted files and switches the administrative system to another system during a planned change of administrative systems.
/etc/umountd.c	This file contains C language source code for **/etc/umountd**.
/etc/remounts	This shell script unmounts the files listed in **/etc/remounts.list** and then tries to mount them again. It is run by the **/etc/umountd** daemon.
/etc/Retry1Mount	This shell script schedules the mounting of $HOME directories. If they cannot be mounted the first time, the mount is scheduled for another time. This program is run by **/etc/SSImounts** when the single-system image configuration starts.

Figure 11-5 (Part 3 of 4). Single-System Image Files

File	Description
/etc/lockfvi.c	This file contains C language source code for a program that, when compiled and run, allows for orderly updating of shared administrative files. To do this, it: 1. Places an advisory lock on the named file. 2. Runs the **vi** editor to allow you to change the file. 3. Removes the lock on the file when you exit the editor.
/etc/Makefile.SSI	This makefile builds **lockfvi**, **unmountd** and **movemailfile** from their source code files. **SSImounts** runs the **make** program with this control file when it starts the single-system image configuration.
/etc/movemail	This shell script makes the **movemailfile** program and then executes it.
/etc/movemailfile.c	This file contains C language source code for a program that, when compiled and run, automatically updates mail spool queues so that mail received while the administrative system is down is moved to the shared spool area.
/usr/adm/ds.msg	This file is a log file that contains messages that describe the progress of configuring the single-system image. It is created when the system starts-up. Use the contents of this file to help determine the cause of any problems that may occur during start up.

Figure 11-5 (Part 4 of 4). Single-System Image Files

Configuring the Administrative System

One system in the network of RTs maintains the files and directories that are mounted by the other systems in the single-system image configuration. Use the following procedure to configure that administrative system.

Configuring the Administrative System

1. Select one of the systems to be the administrative system. The choice ofwhich system to use is arbitrary.
2. Ensure that the **/tmp** file system has at least 300 blocks available.
3. Copy the file containing the single system image sample files to **/tmp**:

   ```
   # cp /usr/lpp/ds/samples/SSI.backup /tmp
   ```
4. Make a working directory in **/tmp** called **SSI** and make it the current directory.
5. Unpack the file to the current directory:

   ```
   # restore -xvf ../SSI.backup
   ```
6. Change the administrative files to match the system.
7. Remove **/tmp/SSI/etc/oug.SSI**
8. Edit **/etc/passwd** to reflect the new path names for user directories.
9. Edit **/tmp/SSI/etc/qconfig** to reflect the printers you want shared.
10. Run the installation shell script:

    # /tmp/SSI/SSIinstall *node*
11. Make a directory, named with the nickname of the administrative system, in the **/u** directory.
12. Move all user login directories to be under this new directory.
13. Shut down the system and restart the system.

More Detailed Information

1. Select one of the systems to be the administrative system. The choice of which system to use is arbitrary.
2. Ensure that the **/tmp** file system has at least 300 blocks available. Use the **df** command to determine the free space on the **/tmp** filesystem. For example:

   ```
   # df /tmp
   Device       Mounted on       total     free  used     ifree  used
   /dev/hd3     /tmp              1856     1844    0%       636    0%
   ```

   The `free` column (here it shows 1844 blocks available) must show at least 300 blocks. If it does not, remove some of the files in the **/tmp** filesystem.

3. Copy the file containing the single-system image sample files to **/tmp**. All required files are contained in **/usr/lpp/ds/samples/SSI.backup**, which is in **backup** format:

```
# cp /usr/lpp/ds/samples/SSI.backup /tmp
```

4. Make a working directory in **/tmp** called **SSI** and make it the current directory:

```
# mkdir /tmp/SSI
# cd /tmp/SSI
```

5. Use the **restore** command to unpack the file to the current directory:

```
# restore -xvf ../SSI.backup
```

   This command extracts the files from **SSI.backup** and places them in the current directory, creating subdirectories as needed.

6. Use your favorite editor to change the administrative files in **/tmp/SSI** subdirectories to match the system. These files are:

   etc/adminserver
   Since this procedure is configuring the administrative system, enter the nickname of the system that you are configuring.

   etc/SSImachines
   Enter a list of the nicknames of all systems in the single-system image configuration. Each nickname in the list begins a new line. The list includes the name of the administrative system.

   usr/adm/user.cfile
   The `udir` parameter in this file defines the default $HOME directory for new users. Change this parameter to:

   `udir` `/u/`*node*/

   In this format *node* is the nickname of the administrative system. If you have changed any of the other default values, such as the default login shell, change those values in this file also.

   etc/server.files
   Enter a list of files that each nonadministrative system mounts from the administrative system. Each file in the list begins a new line. As a minimum, the list must include:

   - **/etc/passwd**
   - **/etc/opasswd**
   - **/etc/group**
   - **/etc/ogroup**
   - **/usr/adm/user.cfile.**

You can add files to this list as you need, but **do not** include the following files:

- **/etc/SSImounts**
- **/etc/umountd**
- **/etc/remounts**

Note:

The remote **write** command uses a file **/etc/wwwmachines** for a similar purpose (to allow writing to users on remote systems - see "Using the write Command with Distributed Services" on page 11-44). For less complex single-system image configurations, **wwwmachines** and **server.files** can be linked to the same file.

etc/server.dirs
Enter a list of directories that each nonadministrative system mounts from the administrative system. Each file in the list begins a new line. If you do not mount any directories, create this file as an empty file:

```
# cat /dev/null > /tmp/SSI/etc/server.dirs
```

etc/remounts.list
Enter a list of files and directories that each nonadministrative system unmounts when the administrative system cannot be contacted. These files are mounted again when the administrative system becomes available. Do **not** include the following files in this list:

- **/etc/opasswd**
- **/etc/ogroup**
- **/usr/adm/user.cfile.**

etc/ug.SSI
This file contains a list of user ID and group ID translations in a form that can be used as input to **dsldxprof**. This list contains a large number of user IDs. You may need to change it to match your system. See "Using the dsldxprof Command" on page 11-121 for information about the format of this file.

7. Remove **/tmp/SSI/etc/oug.SSI**:

```
# rm /tmp/SSI/etc/oug.SSI
```

When single-system image starts, **/etc/SSImounts** compares **/etc/ug** to **/etc/oug**. If **/etc/ug** is different, **SSImounts** updates the Network Users/Groups Table using the translations in **/etc/ug**. Therefore, removing **/etc/oug** causes **/etc/SSImounts** to update the Network Users/Groups Table when the single-system image starts.

8. Edit **/etc/passwd** to reflect the new path names for user directories. When the full system is configured and running, $HOME directories are put into the directory **/u/***node* (*node* is the nickname of the administrative system), instead of **/u**. For example, if **/etc/passwd** contains the following entry and the administrative system is `george`:

   ```
   geo::201:0::/u/geo:/usr/bin/csh
   ```

   Change it the entry to:

   ```
   geo::201:0::/u/george/geo:/usr/bin/csh
   ```

   For a standard format password file that does not use any login programs in the **/u** directory, the following commands change the entry when system `george` is the administrative system:

   ```
   # cp /etc/passwd /etc/opasswd
   # sed "s?/u/?/u/george/?g" /etc/opasswd > /etc/passwd
   ```

   These commands change all occurrences of */u/* in the password file to */u/george/*.

9. Edit **/tmp/SSI/etc/qconfig** to reflect the printers you want shared. This file is supplied to give an example of what the file looks like on an operating single-system image configuration. *Do not use this file as supplied.* Instead, move the file to a backup version and copy your local **qconfig** file to this directory:

   ```
   # mv /tmp/SSI/etc/qconfig /tmp/SSI/etc/qconfig.samp
   # cp /etc/qconfig /tmp/SSI/etc
   ```

   Using **qconfig.samp** as an example, change the new **/tmp/SSI/etc/qconfig** to match your desired configuration. See *AIX Operating System Technical Reference* for the format of the **qconfig** file. **Defining Local Printers**

   Local printers are those printers that are directly attached to a specific system in the single-system image configuration. When a user at that system enters the **print** command without specifying a printer, the output is printed on the local printer. The local printer is a different printer on each system in the single-system image configuration. All printers defined in the **qconfig** file that are *not* defined as shared printers, are local printers. If each system has different types and numbers of local printers, the illusion of a single-system image is not maintained with respect to local printers. In general, local printers use **/usr/lpd/piobe** as the `backend`.

   The following example defines a local printer in the example **qconfig.samp** file:

   ```
   lcl:
           argname = local
           device = dlcl
           discipline = sjn

   dlcl:
   ```

```
        file = /dev/lp0
        backend = /usr/lpd/piobe -device=d3812s2 -profile=/etc/ddi/sprinter
                -statusfile -pname=local
        header = group
        trailer = never
```

Defining Shared Printers

Note: You must have Interface Program for use with TCP/IP installed to define a shared printer.

Shared printers are those printers that are defined in the **qconfig** file as a TCP/IP printer. In most cases these printers use **/usr/bin/lprbe** as a backend and supply the **-pserver** flag to that program. All systems in the single-system image configuration can use shared printers in the same way, that is, by specifying the same printer specification to the **print** command. For example, the following entry from **qconfig.samp** defines a printer that all systems can access with the **print auschs** command:

```
chs:
        argname = auschs
        friend = TRUE
        device = dchs

dchs:
        header = always
        backend = /usr/bin/lprbe  -statusfile -pserver=auschs
```

This entry defines a printer on system `auschs`. Even a user on `auschs` can enter **print auschs** and the output goes to that printer.

See the *Interface Program for use with TCP/IP* book for more information about remote printers.

10. Run the installation shell script:

    # /tmp/SSI/SSIinstall *node*

    In this format, *node* is the nickname of the administrative system that you are configuring. This shell script performs the following functions on the administrative system:

    a. Makes backup copies of the following files:
       - **/etc/rc.ds**
       - **/etc/qconfig**
       - **/usr/adm/user.cfile**

    b. Copies the files in the file tree beneath **/tmp/SSI** to their respective places in the root file system:
       - **/etc/rc.DS**
       - **/etc/adminserver**
       - **/etc/SSImounts**
       - **/etc/server.files**
       - **/etc/server.dirs**
       - **/etc/rc.end**
       - **/etc/movemail**
       - **/etc/remounts**
       - **/etc/ug.SSI**
       - **/etc/Retry1Mount**
       - **/etc/SSImachines**
       - **/etc/newadmin.sh**
       - **/etc/remounts.list**
       - **/etc/qconfig**
       - **/etc/Makefile.SSI**
       - **/etc/rc.ds**
       - **/etc/umountd.c**
       - **/etc/lockfvi.c**
       - **/etc/movemailfile.c**
       - **/usr/adm/user.cfile**
       - **/SSIinstall**
       - **/SSIkeepconfig**
       - **/DS-SSSI.README**

11. Make a directory, named with the nickname of the administrative system, in the **/u** directory. For example, if the administrative system's nickname is `george`:

    ```
    # mkdir /u/george
    ```

12. Move all user login directories to be under this new directory. If you have not changed the way user login directories are assigned, the following shell command string moves the directories (for system `george`):

    ```
    # cd /u
    # cat /etc/opasswd | grep /u/ | cut -f6 -d: | cut -f3 -d/ | xargs -i -t mvdir {} george
    ```

Notice that the file **/etc/opasswd** used as the source of information is the backup copy of your original **/etc/passwd** that you created in Step 8 on page 11-38. If you did not create that file, then use the modified **/etc/passwd** and change the second **cut** command to:

```
cut -f4 -d/
```

13. Shut down the system and restart the system.

Configuring a Client System

Repeat the following procedure on each of the remaining systems in the network to complete the single-system image configuration.

Configuring Other Systems

1. Mount the **/tmp/SSI** directory from the administrative system onto the local **/mnt** directory.
2. Run the **/mnt/SSIinstall** program providing the nickname of the administrative system as a parameter. For example, if the administrative system is `george`:

   ```
   # /mnt/SSIinstall george
   ```

3. Make a directory, named with the nickname of the local system, in the **/u** directory.
4. Move all user login directories to be under this new directory.
5. Shut down the system and restart the system.

More Detailed Information

1. Mount the **/tmp/SSI** directory from the administrative system on the local **/mnt** directory. For example, to mount the files from system `george`:

   ```
   # mount -n george /tmp/SSI /mnt
   ```

2. Run the **/mnt/SSIinstall** program providing the node name of the administrative system as a parameter. For example, if the administrative system is `george`:

   ```
   # /mnt/SSIinstall george
   ```

   This shell script performs the following functions on the local system:

   a. Makes the directory **/tmp/SSI** on the local system and copies the file tree from the mounted directory, **/mnt** to it.

b. Adjusts the ownership and access permissions of files in that file tree.

c. Makes backup copies of the following files:

- **/etc/rc.ds**
- **/etc/qconfig**
- **/usr/adm/user.cfile**

d. Copies the files in the file tree beneath **/tmp/SSI** to their respective places in the root file system:

- **/etc/rc.DS**
- **/etc/adminserver**
- **/etc/SSImounts**
- **/etc/server.files**
- **/etc/server.dirs**
- **/etc/rc.end**
- **/etc/movemail**
- **/etc/remounts**
- **/etc/ug.SSI**
- **/etc/Retry1Mount**
- **/etc/SSImachines**
- **/etc/newadmin.sh**
- **/etc/remounts.list**
- **/etc/qconfig**
- **/etc/Makefile.SSI**
- **/etc/rc.ds**
- **/etc/umountd.c**
- **/etc/lockfvi.c**
- **/etc/movemailfile.c**
- **/usr/adm/user.cfile**
- **/SSIinstall**
- **/SSIkeepconfig**
- **/DS-SSSI.README**

3. Make a directory, named with the nickname of the local system, in the **/u** directory. For example, if the local system's nickname is `zeus`:

```
# mkdir /u/zeus
```

4. Move all user login directories to be under this new directory. If you have not changed the way user login directories are assigned, the following shell command strings move the directories (for system `zeus`):

```
# cd /u
# cat /etc/passwd | grep /u/ | cut -f6 -d: | cut -f3 -d/ | xargs -i -t mvdir {} zeus
```

This command uses **/etc/passwd**, which is still the original file for the client system (unchanged, as yet, for single-system image). If you have already mounted **/etc/passwd** from the administrative system (it happens automatically if you have restarted the system after Step 2 on page 11-41), use the file **/native/etc/passwd** instead.

5. Shut down the system and restart the system.

Changing the Administrative System

Perform the following steps to change the administrative system from one system to another in the single-system image configuration:

1. Edit **/etc/adminserver** to remove the name of the current administrative system and enter the name of the new administrative system.
2. Save the file.
3. Wait a few minutes to allow all systems in the single-system image configuration to update their copies of the file.
4. Shut down and reboot the old administrative system.

Using the write Command with Distributed Services

The **write** command sends a message to another user who is logged in. "Communicating with Another User—The write Command" on page 5-33 describes how to use the **write** command to communicate with other users who are logged into the local node. However, in a Distributed Services environment, **write** can send messages to users logged into remote nodes.

There are two ways to specify the remote nodes to which **write** can send messages:

- Use the **-n** flag to specify a particular node.
- Create a **/etc/wwwmachines** file to list a group of nodes.

To send a message to a user on a particular remote node, use the **-n** flag in a command of the form:

```
write -n dept998 johnson
```

After entering the command, type your message and then press **END OF FILE**. The **write** command sends your message only if user `johnson` is logged in on node `dept998`.

To cause **write** to check multiple remote nodes for the specified logged in user, create a file named **/etc/wwwmachines**. This file is simply a list of remote node nicknames or NIDs, for example:

```
dept998
dept91t
108133Eb
20813AE4
central
```

If **/etc/wwwmachines** exists, the **write** command checks all of the nodes listed for the logged in user. If you use the **-n** flag, **write** does not use **/etc/wwwmachines**.

The **writesrv** program, which is run from **/etc/rc.inlcude** at system start, provides the remote **write** capabilities. For more information, see **write** and **writesrv** in *AIX Operating System Commands Reference.*

Using Distributed Services to Provide Remote Queues

One benefit of having a Distributed Services network is that the systems on the network can share resources. You can configure the queueing system so that you can use printers or other devices on remote systems just as you do those on your local system.

Application programs that can run in a noninteractive way can also be shared in this manner. The **backup** and **restore** commands can be used with the queueing system. This permits you to put the **backup** and **restore** programs on only one machine, and the other systems on the network can share both the software and the tape drive.

This section describes the procedures to configure the Distributed Services network to use the queueing system. Using printers on remote systems is discussed first. This will tell you how to access the queues on a remote system. Then, "Remote Backup and Restore" on page 11-47 discusses the procedures for using the **backup** and **restore** programs with remote queues. This section can be used as a model to assist you in setting up other programs to run under the Distributed Services queueing system. Finally, "Configuration" on page 11-52 discusses important points of the Distributed Services configuration that allow the queueing system to operate properly.

Remote Printers

When you submit a print request, you specify which queue is to handle the request. The queues are defined in **/etc/qconfig**. If you do not specify a queue, the first queue stanza in **/etc/qconfig** is used by default. After submitting the print request, do not remove the node that contains the file(s) to be printed from the network until the printing has occurred.

Defining Printer Queues

To submit a print request to a remote printer, you must define a queue in **/etc/qconfig** on your local system that refers to the queue on the remote system which handles the remote printer. For example, a remote system with node ID 108133eb has a queue named **lpr** that is serviced by two printers.

The **/etc/qconfig** file on the remote system contains the following queue stanzas:

```
lpr:
        argname = -l
        device = lpdev0, lpdev1

lpdev0:
        file = /dev/lp0
        backend = /usr/lpd/lp

lpdev1:
        file = /dev/lp1
        backend = /usr/lpd/lp
```

The first stanza defines the queue **lpr**. The *argname* field tells how to invoke the queue (with the command `print -l` *filename*). The next two stanzas identify the printers that service the queue.

To submit a print request to the queue **lpr** from your system you must have the following queue stanza in your **/etc/qconfig** file:

```
rlp:
        argname = -rl
        node = 108133eb
        rargname = -l
```

This stanza defines the queue **rlp**. The *argname* field tells how to invoke the queue from your system. The *node* field specifies the remote system ID (you could use the remote system nickname instead of the ID), and the *rargname* field specifies the queue for the remote printer.

To spool a file from your system to a remote system, it is recommended that you add the keyword **CD** to the queue **lpr** stanza in your **/etc/config** file. This keyword causes the **qdaemon** to override the default action, change the directory of the spool file, and translate the group and user IDs. For more information on the **CD** keyword, see "Spooling" on page 3-21.

Using Remote Printers

A print command requesting the indirect queue **rlp** in the example above has the form `print -rl` *filename*. The *node* field supplies the remote node ID and the *rargname* field supplies the remote queue *argname*, making it just as simple to print on a remote printer as it is to print on a local printer.

Since the remote queue **lpr** is serviced by two printers, you can specify which remote printer you want to send your job to just like you would with a local queue serviced by two printers. To send your print job to the first printer, use the command:

`print -rl:0` *filename*

To send it to the second printer, use:

`print -rl:1` *filename*

If *rargname* is omitted from an indirect queue stanza, the default queue on the specified node is used.

You can put an *up* field in an indirect queue stanza to control the status of the indirect queue. This affects only the status of the indirect queue, not the status of the actual queue on the remote system. Use this field to restrict print requests from particular nodes while allowing other nodes to submit requests.

An indirect queue may not contain *device*, *discipline*, *acctfile*, or *friend* fields.

Remote Backup and Restore

You can set up the **backup** and **restore** commands to work under the queueing system by running them in the unattended mode. This section describes how to define **backup** and **restore** queues and how to use the **print** command to submit remote backup and restore requests. After submitting the backup or restore request, do not remove the node that contains the file(s) to be operated on from the network until the operation is complete.

Defining Backup and Restore Queues

Configuring queues to perform remote **backup** and **restore** requests is similar to configuring queues for print requests. You must define queues in **/etc/qconfig** on your local system that refer to the queues on the remote system that has the tape drive and the **backup** and **restore** programs.

In the following example, there are three queues on the system installed on system `zeus`. One is for backing up files, one is for restoring files, and one is for writing the names of files on a tape (table of contents) to another file. The **/etc/qconfig** file on the remote system defines the queues named **bbq**, **brq**, and **btq** with the following stanzas:

```
* bbq - background backup queue
bbq:
        argname = -backup
        friend = FALSE
        device = dbq

dbq:
        backend = /bin/sh /usr/lpd/br/bbq

* brq - background restore queue
brq:
        argname = -restore
        friend = FALSE
        device = drq

drq:
        backend = /bin/sh /usr/lpd/br/brq

* btq - background table of contents queue
btq:
        argname = -toc
        friend = FALSE
        transform = TRUE
        device = dtq

dtq:
        backend = /bin/sh /usr/lpd/br/btq
```

The shell scripts specified in the *backend* fields above are included as part of the AIX operating system. By adding these stanzas to **/etc/qconfig**, the **backup** and **restore** commands can be run in the background. Notice the *transform* field in the queue **toc**. This is necessary for the queue to work properly. See "Transformation" on page 11-53 for an explanation of this field.

To submit requests to these queues from another system, you must have the following queue stanzas in **/etc/qconfig** on the other system:

```
* bak - queue for remote backups
bak:
        argname = -backup
        node = zeus
        rargname = -backup

* res - queue for remote restore
res:
        argname = -restore
        node = zeus
        rargname = -restore

* toc  - queue for remote table of contents
toc:
        argname = -toc
        node = zeus
        rargname = -toc
```

Using Remote Backup and Restore

Since each of the queues on both the local and remote systems in the example above are defined to be invoked with the same *argname*, the **print** commands shown in this section can be used on both the local and remote systems. This demonstrates how the queueing system can make using remote facilities just like using local ones.

When **backup** is run in the interactive mode, the names of files to backup are taken from standard input. This is true in the background mode also. To back up specific files, use the **print** command as shown below:

`print -backup`
file1
file2
file3
END OF FILE

Another way to back up files is to use the **find** command as follows:

```
find /u/amy -print | print -backup
```

In this example, the **find** command pipes the names of all files in the directory */u/amy* to the **print** command.

If the tape drive is not ready when the **qdaemon** processes your backup request, your request will be canceled. The **qdaemon** on the system with the tape drive attached to it writes status on each queue request to a file in the directory **/tmp** with the same name as the queue name but with **.out** added to the end. You can see if any requests have been canceled by looking at this file. The **qdaemon** also sends job status in a mail message to

/usr/mail/adm. Refer to "Control of Remote Queues" on page 11-51 to see how to send a message to an operator requesting special preparations for your job.

To restore files, enter the **print** command as shown in the following example:

```
print -restore /u/amy/\*
```

This example restores all files on the tape that are in the directory */u/amy*. The backslash preceding the asterisk is necessary because the asterisk has a special meaning to the shell script that invokes the **restore** command.

To restore all of the files from the backup, use the **ALL** option as shown below:

```
print -restore ALL
```

The queue **toc** writes the contents of a backup tape to a specified file. Use the following command to invoke this queue:

print -toc -T *file*

The specified file must exist before entering this command.

See "Control of Remote Queues" on page 11-51 for how to send operator messages if you cannot prepare the tape drive on a remote system yourself.

Target Node and Qualifying Directory

In a Distributed Services network the **restore** program needs a node ID and directory to identify where to put files. The node ID that **restore** uses is called the *target node*, and the directory is called the *qualifying directory*. The qualifying directory specifies where to put files with directory-relative file names.

The **backup** command puts this information in the header on the backup medium so that the **restore** program can read it. However, older backup formats did not include this information in the headers. You can use the flags **-N** and **-Q** with the **restore** program to specify a target node and qualifying directory if they are not specified in the header. You can also use these flags if you want to override the target node and qualifying directory that are in the header. For example, you can restore files that were backed up from another system. If on node **hera** you did a remote backup from directory */u/brian* of the file **client.data** as:

```
$ print -backup
client.data
END OF FILE
$
```

the **backup** program would write **hera** as the target node, and */u/brian* as the qualifying directory, onto the backup header. You can restore the file to the directory */u/amy* on node **zeus** with the command:

```
$ print -restore -Nzeus -Q/u/amy client.data
$
```

Do not leave any space between the **-N** and **-Q** flags and their arguments.

If you do not use an **-N** or **-Q** flag in a restore request, and the backup header does not have a target node or qualifying directory, then an error occurs and the **restore** program exits. You can use either flag without the other in a restore request.

You can also use the **-N** and **-Q** flags in a backup request. In a backup request, these flags tell where to find the specified files, as well as what to write on the backup header. For example, if you make a backup request on node **mark** as follows:

```
$ print -backup -Ngeorge -Q/tmp < file.list
$
```

the **backup** program would write **qeorge** as the target node, and */tmp* as the qualifying directory, onto the backup header. The files listed in **file.list** are copied from */tmp* on node **george** to the backup medium.

You can use **-Q** in a backup request without **-N**, but if you use **-N** without **-Q**, an error will be generated.

Control of Remote Queues

You can use the four **print** command flags **-rr**, **-dg**, **-ot**, and **-of** to provide limited control of remote nodes in a Distributed Services environment. See the **print** command in *AIX Operating System Commands Reference* for information on these flags. For example, to cause the **qdaemon** on the remote system **charlie** to reread the **/etc/qconfig** on that system, enter the command:

```
print -rr=charlie
```

To bring down the **qdaemon** after all currently running jobs are finished, enter the command:

```
print -dg=charlie
```

In a Distributed Services network a remote system may not be close enough for you to prepare a device for your request. For instance, a print job may require special paper, or a backup or restore job may need a tape drive prepared. Use an **-ot** or **-of** flag to send a message to a remote system console requesting the operator of that system to perform some task and reply to your message.

These flags cause the remote **qdaemon** to display your message when it is ready to process your request. The **qdaemon** then waits for a response to your message. If the response is affirmative, the **qdaemon** performs the request, but if the response is negative or if there is no response within a timeout period, the **qdaemon** cancels the request. The **qdaemon** writes a status message for each request to a file in the directory **/tmp** that has the same name as the queue but with **.out** added to the name. Read this file to check the exit status of your jobs.

As an example, suppose you need to back up a file, but the tape drive and **backup** program are on a remote system. Enter the following command to submit your request to a remote queue and send an operator message to the remote console:

```
find /u/amy/data -print | print -backup -ot="Please load blank tape"
```

The message `Please load blank tape` is sent to the remote console along with instructions on how to use the **write** command to reply to the message (see "Using the write Command with Distributed Services" on page 11-44). After loading a blank tape, the operator uses the **write** command to respond with the message `ok`. The **qdaemon** then processes the request. If the tape drive cannot be prepared for some reason, the operator responds with `cancel`, and the **qdaemon** will cancel the request. If the operator cannot be contacted, the **qdaemon** will cancel the request.

If a longer message is required to explain your request to the operator, you can put the message in a file and use the **-of** flag to send the file. The **print** command will look something like the following:

```
find /u/amy/data -print | print -backup -of=message_file
```

The operator responses are the same as for the **-ot** flag.

Status of Remote Queues

Remote queues have four status conditions that appear when you enter the **print -q** command in addition to the normal states **READY**, **RUNNING**, **WAITING**, and **OFF**. If an indirect queue refers to a queue on a remote system that cannot be contacted, the status of the indirect queue is listed as **NODE DOWN**. If the *rargname* field of an indirect queue does not match the *argname* field of a queue on any remote systems, the queue is listed as **BAD RARG**. If the status file for a queue cannot be opened, the queue is listed as **UNKNOWN**. If the status in the file cannot be understood, the queue is listed as **BAD**.

Configuration

In order for the queueing system to work properly in the Distributed Services environment the following information must be considered carefully.

Permissions

On each system in the network, the program **/bin/print** should be owned by **root**, have a group ID of **printq**, and have an absolute permission mode setting of **6555**. This setting allows all users to execute the **print** command, and sets the effective group ID for all users to **printq** so that network translations (discussed below) can allow proper permission for all remote **qdaemons**. Each system's **/etc/qdaemon** program should be owned by root, have a group ID of **printq**, and have an absolute permission mode setting of **6554**. If it is necessary to restart a **qdaemon**, the user must have superuser authority or be a member of the **printq** group, since **qdaemon** is executable only for them.

ID Translation

Each node on the network should map the **printq** group ID to the same outbound network ID and map that inbound network ID back to the **printq** group ID. This permits the **print** command to submit a request to any remote **qdaemon** and to check the status of remote queues and devices.

In order for a user to access a remote file, the node that contains the file must have an inbound translation for the user's outbound network ID. The translated inbound ID must have sufficient permission to access the file.

Transformation

When a user specifies a file name in the **print** command, the file name is passed to the program designated by the *backend* field in **/etc/qconfig**. If the user wants to send the file to a printer, the file name must be *transformed* to include the node ID and directory where the file actually is. The transformation is necessary because the path name that the user provides to the **print** command is relative to the file systems on his/her local node. However, the structure of the file system on the node that performs the backup or restore may be different, so that the specified path name must be changed to one that is meaningful to the remote node.

A *transform* field in **/etc/qconfig** tells **print** whether or not to transform the file name. If the backend program for a queue is friendly (that is, if the *friend* field is not present or is set to *TRUE*), then the default value for *transform* is *TRUE*. The default value for *transform* with unfriendly backends is *FALSE*.

The backend programs for the **backup**, **restore**, and **toc** queues are unfriendly because they do not follow special conventions for communicating with the **qdaemon**. The **toc** queue has a *transform* field that is set to **TRUE**, overriding the default, so that the file name specified in the **print** command is transformed. This is necessary because the backend needs to know the actual node and directory of the file. The **restore** queue backend does not need the file names to be transformed since these names refer to data on the backup medium, not on a file system. The target node and qualifying directory information identifying where to put the data is read from the tape or passed to the **restore** queue backend from the **print** command's **-N** and **-Q** flags (see "Target Node and Qualifying Directory" on page 11-50).

Using Distributed Services to Provide Code Service

Users of many or all of the machines in a network often require the same licensed programs. To install and maintain copies of the licensed programs on each machine is inefficient: each machine must devote the same amount of disk space to storing identical copies of the same licensed programs, and the person responsible for system maintenance must perform each software installation or update repeatedly, once for each machine in the network. ***Code service***, the process of giving client systems access to the programs stored on a server system, can reduce or eliminate these inefficiencies.

An ***active code service*** client, which has relatively few programs installed, connects to the server at system start and can then run programs installed on the server. A passive code service client uses the server as a source for programs (instead of diskettes, for example), but actually runs its own copy of the programs.

The general procedure for providing active code service is to install licensed programs on a server and then, using the remote mount capabilities of Distributed Services, mount several directories from the server over those same directories on the client. For example, the client mounts **/usr/bin** from the server over its own **/usr/bin**. After this mount, a program installed in **/usr/bin** on the server is available to the client, even though **/usr/bin** on the client does not contain the program.

To connect itself to a server for active code service, a client runs the **chngstate** command, normally from **/etc/rc** at system start. In a simple case, the connection requires only a series of remote mounts. The client mounts the following server directories over its own directories of the same name:

/usr/bin
/usr/database
/usr/datamgt
/usr/dos/bin
/usr/games
/usr/include
/usr/lib
/usr/lpp
/usr/man
/usr/pub

Thus, the clients conserve disk space, since only the server must install the licensed program. An additional advantage of active code service is that, in most cases, only the server must install new programs or update existing ones.

Note: A client can have an active code service attachment to only one server at any given time.

Although some active code service connections are more complex than others (that is, they involve more than the ten remote directory mounts), AIX can complete most connections automatically. For example, because it contains information unique to a particular

machine, each machine must have access to its own **/etc** directory. Therefore, a client cannot mount **/etc** from the server over its own **/etc** directory. Similarly, some licensed programs must install device drivers into the kernel of each machine they run on. So that they will be able to run in an active code service environment, licensed programs with such special requirements provide a ***program subset*** (the files and data that must be installed on each active code service client).

Typically, program subsets are small and only a few licensed programs require them. Thus, they do not significantly increase the requirement for disk storage on the client systems. However, they must be installed and updated on each client machine. AIX provides facilities for performing these system maintenance tasks automatically. When you install or update a licensed program on the server that has a program subset, the client can determine what program was installed or updated, install the appropriate program subset, and then update the appropriate program subset, if necessary. The commands involved in this process are:

chngstate Controls the connection process.

chkcomp Detects newly installed or updated programs on the server, determines whether they require program subsets, and checks code version, release, and level compatibility between the server and the active code service client. The **chngstate** process runs **chkcomp**.

installc Installs licensed programs and program subsets on the active code service client. The **chngstate** process runs **installc**.

updatec Updates licensed programs and program subsets on the active code service client. The **chngstate** process runs **updatec**.

Thus, while program subsets must be installed on every client, they only have to be restored once from diskette or tape (at the server). The server does not install program subsets.

To run properly, some licensed programs require that parts of AIX or other programs be installed at a particular version, release, and level. Often, when the client and the server are incompatible (for example, after the server has applied an update to a licensed program), the AIX code service programs can automatically upgrade the client and then complete the connection of the client to the server. In some cases, the person responsible for software maintenance on the machines may have to intervene.

This section explains how to use AIX and Distributed Services to provide code service in an IBM RT network. "Providing Active Code Service" on page 11-56 and "Maintaining an Active Code Service Network" on page 11-66 deal primarily with active code service. "Providing Passive Code Service" on page 11-79 covers topics that are unique to passive code service.

Notes:

1. Before you begin to configure a code service environment, the hardware to support Distributed Services should be installed in your network and you should be familiar with the information in the previous sections of this chapter, especially "Configuring a Basic Distributed Services System" on page 11-17. If your network is already configured for Distributed Services, you may not have to perform all of the tasks described in this section.
2. Code service makes it possible for you to use a single copy of a licensed program on multiple systems. Before you use code service, however, it is your responsibility to ensure that your license agreement for each licensed program allows you to distribute that licensed program to multiple systems.

Providing Active Code Service

Code service requires the same hardware and software that Distributed Services requires. However, to provide active code service, you must customize the software specifically for that purpose. Following are the general steps in customizing software to provide active code service:

Customizing Software for Active Code Service

1. Install the support programs on the server. (See "Installing the Support Programs on the Server" on page 11-57.)
2. Back up the VRM from the server. (See "Backing up the VRM from the Server" on page 11-57.)
3. Customize the server. (See "Customizing the Server" on page 11-58.)
4. Back up the server's files. (See "Backing up Files to be Installed on Clients" on page 11-58.)
5. Install other licensed programs on the server. (See "Installing Additional Licensed Programs on the Server" on page 11-59.)
6. Create the first active code service client. (See "Installing Programs on the First Active Code Service Client" on page 11-61.)
7. Customize the first active code service client. (See "Customizing the First Active Code Service Client" on page 11-62.)
8. Install software from the first client on the remaining clients. (See "Installing Software on the Remaining Clients" on page 11-65.)

Installing the Support Programs on the Server

Install the following support programs on the server from AIX and licensed program diskettes. (For information about installing these programs, see *Installing and Customizing the AIX Operating System.*)

- IBM RT AIX Operating System, with:
  - A **/tmp** minidisk size of 35000 blocks
  - A **/dump** minidisk size of 10000 blocks
  - Adequate free space to allocate later for the minidisks that contain program copy files (as is explained under "Installing Additional Licensed Programs on the Server" on page 11-59)
- IBM RT Systems Network Architecture (SNA) Services licensed program
- IBM RT Distributed Services licensed program (purchased separately)
- VRM Baseband Adapter Device Driver or IBM Token-Ring Adapter Device Driver from the IBM RT Virtual Resource Manager Device Driver diskette.

Note: For active code service, the support programs on the client must be at the same version, release, and level that they are on the server.

Backing up the VRM from the Server

Note: This step is optional, but advised if you:

- Install a version of the VRM that has updates.
- Have a relatively large number of client systems to install.
- Do not have a backup copy of the VRM currently installed on the server.

With AIX running, use the **cvid** command to back up the server's VRM. Enter a command of the form:

```
cvid /dev/fd0
```

Insert a formatted 1.2M byte diskette into the first diskette drive when the **cvid** command prompts you to do so. The **cvid** command takes approximately 30 minutes to run and the VRM backup requires at least two 1.2M byte diskettes.

Customizing the Server

To customize a server for active code service:

1. Use the **ugtable** command to provide the following Distributed Services UID and GID translations:

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
guest	U	100	-	*	*
usr	G	100	-	*	*

   These translations permit clients to perform the required active code service remote mounts.

2. Edit the **/etc/rc** file to remove the comment symbol (#) from the first position of the line that contains the **chngstate** command. This line is near the end of the file.

3. Use the **devices** command to add the following devices to the system, if appropriate:

   a. `tape`

      Add the `IBM RT Streaming Tape Drive` device if a tape adapter and tape drive are installed on the server. You can use tapes to install the support programs on the clients if the clients have tape drives installed.

   b. `net` or `token`

      If your entire network uses the IBM RT Baseband Adapter for use with Ethernet, add the `net` adapter to the configuration of the server. If your entire network uses the IBM Token-Ring Network RT Adapter, add the `token` adapter. If your network uses a mixture of the two adapters, bypass this step, and then add the appropriate adapter to the configuration of each client.

Backing up Files to be Installed on Clients

To back up the files from the customized server:

1. Unmount the VRM minidisk with the command:

   ```
   umount /vrm
   ```

   Also unmount the **/u** directory if it contains any files or directories that you do not want to install on each client.

2. Change to the / (root) directory (**cd** /).

3. Run the **backup** command appropriate for the medium that you will use to distribute the support programs from the customized server to the first client:

   - To use a streaming tape drive, enter the command:

     ```
     find . -print | backup -ivr -f/dev/rmt0 -C2000
     ```

     The **-r** flag causes volume mounting messages to be suppressed, and **-C2000** sets size (in blocks) of each transfer of data to the tape.

   - To use a diskette drive, enter the command:

     ```
     find . -print | backup -iv -f/dev/rfd0
     ```

     Note: This step is optional. You can also use the AIX distribution diskettes.

Installing Additional Licensed Programs on the Server

Installing the additional licensed programs on the server involves both customizing and actual program installation.

Installing Licensed Programs on the Server

1. Create the following directories or minidisks (see "Creating /usr/lpp.install and /usr/lpp.update."):

   /usr/lpp.install
   /usr/lpp.update

2. Increase the value of the of the shell variable, **ulimit**.
3. Install licensed programs (**installp -b**) or create program copy files for them (**bffcreate**).
4. Run the **setrdperm** command to check permissions on server directories and files.
5. Customize node table on the server (**ndtable**). (This is not part of the licensed program installation process, but should be done at this point in code service configuration.)

Creating /usr/lpp.install and /usr/lpp.update.

The **/usr/lpp.install** and **/usr/lpp.update** directories store program copy files. The amount of storage required varies with the number and size of licensed programs that the server will provide to clients. You may find it convenient to create separate minidisks for these two directories (with the **minidisks** command). You can adjust the sizes of separate minidisks as necessary without having to change the size of the parent file system (or minidisk), **/usr**.

Increasing the Value of ulimit

The shell variable, **ulimit**, sets the maximum size (in blocks) of a file that processes associated with a particular UID can open. Due to the size of many program copy files, you should increase the value of **ulimit** for the user **root** on the server. While the required **ulimit** value varies among active code service environments, 50,000 blocks is usually adequate. To set the value of **ulimit**, use the **adduser** command to change the **Filesize** field for **root** to the new value (for example, 50000). To reset the value of **ulimit** for the current login session only, run **ulimit** from the command line:

ulimit 50000

Installing Licensed Programs and Creating Program Copy Files

In most cases, you will install licensed programs on the server (so that the server can run them) as well as create program copy files for them. Clients, in turn, use the program copy files (which are in backup format) as their source for installing program subsets or complete programs. To install licensed programs and create program copy files, use the **installp -b** command. The **-b** flag causes **installp** to create a program copy file (by calling the **bffcreate** command) and then use the program copy file to install the licensed program on the server. **bffcreate** stores the program copy files for installing programs in the **/usr/lpp.install** directory and the program copy files for updating programs in the **/usr/lpp.update** directory.

If you want only a program copy file for a particular licensed program (that is, you do not want to install and run the licensed program on the server), use the **bffcreate** command. For more information, see "Background for Code Service System Maintenance" on page 11-67 and **bffcreate** in *AIX Operating System Commands Reference.*

Note: The **bffcreate** command processes only files that are in backup format (that is, files that can also be processed by either **installp** or **restore -x**). **bffcreate** does not process files that are other formats (**tar**, for example).

Running the setrdperm Command

After you install the additional licensed programs, run the **setrdperm** command to check the permissions on all server directories and files that the active code service clients will mount. All of these directories and files must have their permissions set to allow **read** by **other.** The **setrdperm** command scans these directories and files for inappropriate permissions. If it finds any, **setrdperm** displays a message and prompt of the form:

```
-r-xr-x---      root      system      /usr/lib/edconfig

< chmod ... /usr/lib/edconfig >? _
```

To make the appropriate changes, enter `y`; **setrdperm** runs the **chmod o+r** command. If you do not want to change the permissions on a particular file, enter `n`.

Customizing the Server's Node Table

To define the clients for the server, run the **ndtable** command. For information about **ndtable**, see "Working with the Distributed Network Node Table" on page 11-96.

Installing Programs on the First Active Code Service Client

To create the first active code service client:

1. Install VRM (from either the diskettes created on the server with the **cvid** command or, if you did not run **cvid**, the VRM distribution diskettes). To do this, insert the first diskette in the first diskette drive and start the system. Provide appropriate responses when the VRM installation process prompts you for input.
2. After VRM is installed, start the system from the Installation and Maintenance diskette.
3. (Optional step) Determine the current and recommended minidisk sizes and change them, if appropriate. For example, the **/usr** minidisk on clients may not have to be as large as the one on the server.
4. Select the option for installing the operating system.
5. Select the appropriate installation medium, `tape` or `diskette`, depending upon the medium you used to back up the files from the server (see 3 on page 11-59), and install the backup copy of the server files.
6. Start the operating system from the fixed disk. (It is normal to receive certain error messages at this point. Their cause will be corrected after the automatic code service installation and updating process have completed.)
7. Run the **ndtable** command to define the server node for the client.
8. Install any licensed programs that clients will require when they are not running in active code service. Among the programs that you may choose to install are text editors, terminal emulators, and data communications facilities. These programs should be available in the form of program copy files on the server. To install them:
    a. Create a temporary directory on the client (for example, **/usr/getcode**).
    b. Mount the **/usr/lpp.install** directory from the server over the temporary directory on the client with a command of the form:

    ```
    mount -n zeus /usr/lpp.install /usr/getcode
    ```

c. Install each program with a command of the form:

```
installp -d /usr/getcode/em78
```

d. Unmount the server directory.

If there are updates for these programs, use the same process for installing them except the command is **updatep**:

Customizing the First Active Code Service Client

To customize the first active code service client, you must configure the client's Distributed Services node, group, and user data, and edit the code service attribute file to identify the active code service server.

Notes:

1. At system start, clients running in **auto** mode will automatically upgrade their programs to be compatible with those on the server, if necessary. Depending upon the number and nature of server upgrades, a client could restart more than once before it completes its attachment to the server.
2. Do not create separate minidisks on clients for any of the directories required for active code service (the directories listed in **/etc/codeserve/csomdlist**). Such minidisks alter the **/etc/filesystems** information required for remote mounts, making active code service impossible.

Configuring Distributed Services

First, use the **ndtable** command to add the server node to the client's Distributed Services node table. For example:

Remote Nickname	Remote Node ID	Node Security	Data Link Type	Connection Profile	Attachment Profile
server	108133EB	None	Ethernet	108133EB	108133EB
client	2081064A	None	Ethernet	CDEFAULTCFD	CDEFAULT

Note: These are the nodes required for code service. There may be other nodes defined for other Distributed Services purposes.

Then, use the **ugtable** command to provide the following UID and GID translations on the client (these translations are required for active code service):

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
root	U	0		0	server
bin	G	2		2	server
bin	U	2		2	server
sys	G	3		3	server
sys	U	3		3	server
adm	G	4		4	server
adm	U	4		4	server
system	G	0		0	server
staff	G	1		1	server
mail	G	6		6	server
usr	G	100		*	*
guest	U	100		*	*

Note: In some cases, there may be other UIDs or GIDs that require special Distributed Services translations. Any UID or GID on the server with a value of less than 100 should translate to the same UID or GID on the client.

For information about using the **ndtable** command, see "Working with the Distributed Network Node Table" on page 11-96. For information about using the **ugtable** command, see "Working with the Network Users/Groups Table" on page 11-107.

Editing the Code Service Attribute File

The code service attribute file, **/etc/codeserve/serverattach**, specifies the primary server and any backup servers that the active code service client can use. The **chngstate** command, which is run from **/etc/rc**, processes **serverattach**. Ordinarily, **chngstate** begins processing **serverattach** with the first stanza after the `default` stanza. If the client cannot attach to the server described by the first stanza within the specified amount of time, then **changstate** processes the second stanza, and so on through the file. If the client is unable to attach to any of the possible servers, then **chngstate** processes the last stanza in the file, which causes the client to start up in the **stand-alone** state.

Notes:

1. If an attachment fails for any reason other than timing out, **chngstate** does not process any remaining server stanzas. The client starts in the stand-alone state.
2. A client can have an active code service attachment to only one server at any given time.

The **serverattach** file included with AIX contains some example stanzas, similar to the following:

Note: In the example **serverattach** file, these stanzas are commented out (that is, **chngstate** will not process them because the first character on each line is an asterisk). To make these stanzas usable, you must remove the comment symbols from the lines that **chngstate** processes.

```
*   Default system stanza
default:
        state = stand-alone
        mode = manual
        time = 60
        interval = 10

*   Primary server (NOTE: time will default)
server:
        mode = auto
        state = active
        interval = 15

*   Alternate server (NOTE: mode and interval will default)
altserve:
        state = active
        time = 90

*   Standalone
stand-alone:
        state = stand-alone
```

A client with a **serverattach** file like the one in the example first tries to attach to `server`. If that attachment fails (times out), the client tries to attach to `altserv`. If that attachment also fails, the client starts in the stand-alone state.

Note: If an attachment fails for any reason other than timing out, the client starts in the stand-alone state without processing any additional server stanzas. The attributes in **serverattach** are:

mode Indicates how the client should attempt to achieve compatibility with the server. If the value is **auto**, the client will attempt to make its programs compatible (version, release, and level) with those installed on the server without user interaction. If the value is **manual**, the user must perform each client program upgrade manually.

state Indicates the state in which the client should start: **active** or **stand-alone**

time The total amount of time that the client will attempt to attach to the server. The value of **time** is some number of seconds. The client attempts attachment for at least 60 seconds, regardless of the value specified for **time**. If the client cannot attach to the server in the specified time, then **chngstate** processes the next **serverattach** stanza.

interval The amount of time (specified in seconds) that the client delays between attempts to attach to the server. For example, if the value of `time` is `180` and the value of `interval` is `60`, the client will make three attempts to attach to the server identified by this stanza before moving to the next one.

For information about the flags that modify how **chngstate** processes **serverattach**, see **chngstate** in *AIX Operating System Commands Reference.*

If you did not define a network device as part of customizing the server (3b on page 11-58), and all clients will use the same network adapter, run the **devices** command at this point to configure the VRM Baseband Adapter Device Driver or IBM Token-Ring Adapter Device Driver, as appropriate. If your network device is the IBM Token-Ring Adapter Device Driver, you also must remove the comment symbols from the lines in **/etc/rc.ds** that start the IBM Token-Ring Adapter Device Driver

The first client is now ready to attach to the server for active code service. To make the attachment, either run **chngstate** from the command line, or shut down and IPL the client.

Installing Software on the Remaining Clients

To install software on the remaining active code service clients:

1. Back up the files from the first client using the same procedure that you used to back up the files from the server (described under "Backing up Files to be Installed on Clients" on page 11-58).
2. Install the VRM on each client from the diskettes created on the server with the **cvid** command or the VRM distribution diskettes.

3. Create the necessary minidisks and install the files that you backed up from the first client on each of the remaining clients, using the same procedure that you used to install the files from the server on the first client (described under "Installing Programs on the First Active Code Service Client" on page 11-61).

   If you did not define a network device as part of customizing the server (3b on page 11-58) or the first client, run the **devices** command on each client to configure the VRM Baseband Adapter Device Driver or IBM Token-Ring Adapter Device Driver, as appropriate, and then run the **ndtable** command.

This step concludes the process of creating a basic active code service network. To attach the remaining clients to the server for active code service, either run **chngstate** from the command line, or shut down and IPL each client.

Note: Do not create separate minidisks on clients for any of the directories required for active code service (the directories listed in **/etc/codeserve/csomdlist**). Such minidisks alter the **/etc/filesystems** information required for remote mounts, making active code service impossible.

Maintaining an Active Code Service Network

Once you have configured your network to provide active code service, AIX can automatically perform routine maintenance tasks, if the mode of the client is **auto**. For example, if a program subset is already installed on a client and you upgrade the program subset on the server (install a new version or update the existing one), AIX automatically updates the program subset on the client the next time the client attaches to the server. However, there may be instances in which you need to perform some system maintenance tasks manually, or there may be modifications that you need to make to your code service configuration to adapt it to new requirements. This section contains information to help you meet these special requirements.

Using Passive Code Service to Upgrade Active Code Service Clients

When an active code service client cannot automatically upgrade its programs to be compatible (version, release, and level) with those on the server, you must manually upgrade the client. Generally, you can use the procedure described in "Providing Passive Code Service" on page 11-79. The following table shows the options for installing updates or new releases of support programs in a code service environment.

	Install	Update
VRM	1	2, 4
Kernel	1, 5	2, 4
SNA Services	1, 3, 5	2, 4
Distributed Services	1, 3, 5	2, 4
VRM Baseband Adapter Device Driver or IBM Token-Ring Adapter Device Driver	1, 3, 5	2, 4

Figure 11-6. Methods for Installing and Updating the Support Programs

Notes:

1. Install from diskette (**VRM**, **Installation and Maintenance**, or **installp**).
2. Update from diskette (**updatep**).
3. Install from the program copy file on the server (**installp**). (This works only after an initial installation has been performed, since it depends upon some of the support programs.)
4. Update from the program copy file on the server (**updatep**).
5. Install from backup diskettes or tapes (Installation and Maintenance diskette).

Background for Code Service System Maintenance

This section presents brief descriptions of the AIX commands and files involved in providing code service:

Commands

bffcreate

The command that creates a program copy file from the specified distribution medium (for example, diskettes). Both **installp -b** and **updatep -b** use **bffcreate** to create program copy files. However, if you only want to create program copy files (that is, you do not want to actually install or update the program on the server), you can run **bffcreate** from the command line.

Note: The **bffcreate** command processes only files that are in backup format. It will not, for example, process files that are in **tar** format.

The **bffcreate** command stores program copy files for complete programs and program subsets in the **/usr/lpp.install** directory. The format for a file name in **/usr/lpp.install** is *name.vvrr*, where *name* is the 8-character licensed program or

program subset name, *vv* is the version number, and *rr* is the release number. (If you use **bffcreate** to process a file that cannot be processed by **installp** or **updatep**, use the **-f** flag to specify a name for the program copy file that follows the format described here.)

The **bffcreate** command stores program copy files for updates in the **/usr/lpp.update** directory. The default format for a file name in **/usr/lpp.update** is `updt`.*yyddd.nnn*, where *yyddd* is the Julian date, and *nnn* is a sequence number for program copy files created that day. You can specify different names for update program copy files by using the **-f** flag.

chkcomp
A command that checks compatibility between a code server and an active code service client. The **chngstate** process runs **chkcomp**, which creates an output file named **/etc/codeserve/cs.compat**.

chngstate
A command that processes the code service attribute file, **/etc/codeserve/serverattach**. **chngstate** controls the attachment of a client to a server and, depending upon how the code service attribute file is customized, can perform most program installations or updates required to achieve compatibility between the server and the client. **chngstate** is usually run from **/etc/rc**, but can be run from the command line.

installc
The command that installs complete programs and program subsets on active code service clients. **installc** is run by the **chngstate** command, not from the command line. **installc** uses the **/etc/codeserve/cs.compat** file as input.

installp
The command generally used to install AIX programs. The **-b** flag causes **installp** to create a program copy file in **/usr/lpp.install** (by calling **bffcreate**) and then uses the program copy file as the medium for installing the program.

setrdperm
A command, to be run on the server, that scans all files contained in the directories listed in **/etc/codeserve/csomdlist** for proper permission settings. If it finds incorrect permissions (that is, not **read** by **other**), **setrdperm** prompts the user before using **chmod** to reset them.

updatec
The command that updates complete programs and program subsets on active code service clients. **updatec** is run (indirectly) by the **chngstate** command, not from the command line. **updatec** uses the **/etc/codeserve/cs.compat** file as input.

updatep

The command generally used to update AIX programs. The **-b** flag causes **updatep** to create program copy files in the **/usr/lpp.update** directory (by calling **bffcreate**) and then uses the program copy files as the medium for the update.

uucp

A command that copies files from one AIX system to another AIX system. In the Code Service environment, **uucp** will not work unless the Distributed Services ngtable contains entries for `uid uucp (5)` and `gid uucp (5)`..

Files

/etc/codeserve/cs.compat
A file created by the **chkcomp** command that contains information about compatibility between the server and client. For a description of the **cs.compat** file, see "Format of the cs.compat File" on page 11-76.

/etc/codeserve/csomdlist
A file that lists the server directories that active code service clients must mount when they attach to the server. The **setrdperm** command scans the directories listed in **csomdlist**.

/etc/codeserve/serverattach
The attribute file that names all possible servers for a particular active code service client. The **chngstate** command processes this file. When you create a code service network, you must customize this file. (See "Customizing the First Active Code Service Client" on page 11-62 for information about customizing **serverattach**.)

/etc/rc.actvsrvc
Performs the operations necessary to run a system as an active code service client, including mounting remote files.

/etc/rc.include
Starts certain processes that are required by both active code service clients and systems running in the stand-alone state.

Note: If you want to make additions to the commands that are run automatically at system start (for either active code service or stand-alone operation), make them in **/etc/rc.include** rather than in **/etc/rc**.

/etc/rc.stand-alone
Can be used to perform operations necessary to run a system in the stand-alone state, including starting daemon processes. This file can be customized to run programs that are installed only on the client. The **/etc/rc.stand-alone** file shipped with AIX does not start any programs.

/etc/rc.unactvsrvc

Performs the operations necessary to remove a client from active code service, including unmounting remote directories. The **chngstate** command processes **/etc/rc.unactvsrvc**.

Program History Data

A program may be in one of the following states, depending on what type of update actions have occurred:

committed — The normal state for a program. For example, a program is in a committed state immediately after it is installed (assuming an update has not been applied). The committed state implies that the user cannot return to a previous level without starting over with a new installation. The **updatep -c** command commits a previously applied update. The **updatep -ac** command applies and commits an update in the same step.

applied — A successful update has been applied to the program. However, the update can be rejected with the **updatep -r** command, and the program will return to its previous committed state.

unknown — An error has occurred during the update process, and it is impossible to know whether the program is properly installed. Since it is impossible to determine whether any files have been replaced, the installation must be started again.

manual reject — A user requested a manual reject (**updatep -rx**). Generally, manual rejects should not be performed in an active code service environment. If they occur, the program is treated as if it is in the **unknown** state.

The following information records (in the **/usr/lpp/***programname***/lpp.hist** file) indicate the present update state of a program, and they are checked during active code service compatibility checking:

e — Indicates that an error has occurred and that it is impossible to determine whether the program is usable without being reinstalled (that is, it is in an **unknown state)**. The presence of this entry supersedes any other state.

m — Indicates a manual reject. In an active code service environment, this state is handled in the same way that an error (**e**) entry is.

c — Committed. If the last information entry is a **c**, the intallation or update state is committed, and this entry determines the level of the program.

a — Applied. If the last information entry is an **a**, the state is applied, and this entry determines the level of the program

r — Rejected. If the last information entry is an **r**, the state is committed and the last **c** entry determines the level of the program.

An active code service client cannot attach to a server unless its programs are compatible (version, release, and level) with the same programs installed on the server. Any program subsets installed on the client must be the ones associated with the current complete program on the server, as identified in the server's history file.

Ordinarily, an active code service client that runs in **auto** mode (as specified in its **/etc/codeserve/serverattach** file) can perform any necessary licensed program installations or updates without intervention from the user. However, if a client attaches to a server in **manual** mode, or if there are incompatibilities that the code service utilities cannot resolve (specifically, support programs incompatibilities, or situations in which the client is in a later update state than the server is), the user must intervene manually.

The **/etc/codeserve/cs.compat** file records details about the incompatibility. The following table shows the actions that must be taken to make a client compatible with a server under different conditions:

Notes:

1. If a licensed program that requires a program subset is installed on the server, the client must install the licensed program or the program subset to be compatible with the server. Clients running in **auto** mode automatically install the program subset unless the licensed program (complete program) is already installed.

2. In the table, the phrase `Manual intervention required` means that the necessary action cannot be performed automatically by the AIX code service commands. For information on using passive code service to resolve incompatibilities, see “Providing Passive Code Service” on page 11-79.

Server		**Client**	**Action**
version	>	version	Install at client from program copy file.
version	=	version	Compatible. Check release values.
version	<	version	Manual intervention required. Install on the client the same (lower) version that is installed on the server.
release	>	release	Install at client from program copy file.
release	=	release	Compatible. Check update level values.
release	<	release	Manual intervention required. Install on the client the same (lower) release that is installed on the server.

Server Update State	Server Level		Client Level	Client Update State	Action
committed	level	>	level	committed	**update -ac** at client from program copy file.
committed	level	=	level	committed	Compatible.
committed	level	<	level	committed	Manual intervention required. Install on the client the same (lower) level that is installed on the server, then run **chngstate**.
applied	level	>	level	committed	**update -a** at client from program copy file
applied	level	=	level	committed	Compatible.
applied	level	<	level	committed	Manual intervention required. Install on the client the same level that is installed on the server and then run **chngstate** to apply the updates.
committed	level	>	level	applied	1. **update -c** to commit current client level. 2. **update -ac** to apply and commit new client level from program copy file
committed	level	=	level	applied	Compatible

Server Update State	Server Level		Client Level	Client Update State	Action
committed	level	<	level	applied	Manual intervention required. At the client, run: 1. **update -r** 2. **update -ac** If the client level is still higher than the server's, install the same level on the client that is installed on the server. See additional conditions listed below.
applied	level	>	level	applied	1. **update -c** to commit current client level. 2. **update -a** to apply new client level from program copy file
applied	level	=	level	applied	Compatible
applied	level	<	level	applied	Manual intervention required. At the client, run: 1. **update -r** 2. **update -a** See additional conditions listed below.
unknown	level	?	level	?	Manual intervention required. Install the same level on the client that is installed on the server.
?	level	?	level	unknown	Manual intervention required. Install on the client the same level that is installed on the server.

The following conditions also affect compatibility checking and program upgrading:

- If the update state of a program is applied, **installp** cannot install the program. However, **installc** can install the program on a client.

- If the client state is applied, the server state is either applied or committed, and the level of the client is higher than the level of the server, then run **update -r** on the client to attempt to achieve compatibility. After the reject:
  - If the server's level is higher than the client's level, the client performs an **update -a** from a program copy file if the server's state is applied or an **update -ac** if the server's state is committed. If the server and client still are not compatible, manual intervention is required.
  - If the client's version, release, or level is higher than that of the server, manual intervention is required.

Following are examples of program history files:

- History file for a complete program installed on a server:

```
c INed          02.05.0000 060188 root     Version 2.1.1
t INed Program
```

  The c entry identifies committed version, release, and level. The t entry shows the title of the licensed program.

- History file for a complete program installed and updated on a server and requiring a program subset to be installed and updated on an active code service client:

```
c tcpip         02.05.0000 190688 root     Version 2.1.1
t TCP/IP
a tcpip         02.05.0010 220688 root
p cp_tcpip      02.05.0000
c tcpip         02.05.0010 220688 root     Version 2.1.2
```

  The p entry identifies the program subset level required for a particular update level of the complete program. In this example, program subset level 02.05.0000 (version 02, release 05, level 0000) is required for compatibility with a complete program of update level 02.05.0010.

- History file on a client for a program subset:

```
c cp_tcpip      02.05.0000  190688 root  Version 2.1.2
t Client Partial for TCP/IP
f tcpip
```

  The f entry identifies the complete program on the server that is associated with this program subset.

- History file with a program copy file entry:

```
c vied         02.05.0000 190988
t vi Editor
b osplus.02.05
```

  The b entry identifies the program copy file that was used to install this program. A b entry is created only when the name of the program copy file does not match the name of the licensed program. In this example, the name of the program copy file is osplus.02.05, while the name of the licensed program is vied.

- A listing of the committed version history:

```
INed   Version 2.1.2
  IX00975 INed - Underline does not work on vt100
  IX01005 INed - Del Key vt22- does not work
TCP/IP  Version 2.1.1
  IX00995 tcpip - tm to vax hanges virtual terminal
  IX001115 tcpip - ..
```

A header line is provided for each committed program version, identifying both the program and the version applied. The version number is identical to the version number provided on the update diskettes. Below the header line is a list of the problems or adjustments included in the committed update.

To generate this list enter `updatep -A`. If you want to see the history of a specific program include the name of the program, for example:

```
updatep -A ined
```

Format of the cs.compat File

The **/etc/codeserve/cs.compat** file describes any program incompatibilities between an active code service server and one of its clients. The **chngstate** command creates a **cs.compat** file at IPL or when **chngstate** is run from the command line (to change servers).

If manual intervention is required, you can correct the incompatibility between the server and the client with the **installp** and **updatep** commands. If manual intervention is not required, the active code service commands perform the necessary installation and updating of programs on the client (if the stanza for this server in the client's **serverattach** file specifies the mode for the server as **auto**).

Following is an example of a **cs.compat** file:

```
inst n c > c vied 02 05 0000 osplus.02.05 0000
```

The fields are separated by one or more blanks. Following are brief descriptions of their purpose and possible values:

type

Indicates the action necessary to achieve compatibility. Possible values are:

down	Client program is a later version, release, or level than the server program. Manual intervention is required.
inst	Client must install the program.
updt	Client must update the program.
unkn	Server or client program is in an unknown state. The program must be reinstalled.

support programs
: Indicates the program is part of the support programs required for active code service. Possible values are:

y	Program is part of the support programs. Manual intervention is required.
n	Program is not part of the support programs.

server state
: Indicates the current state of the program on the server. Possible value are:

a	Applied.
c	Committed.
u	Unknown (or error). Manual intervention is required. The program must be reinstalled.

compatibility
: Indicates compatibility or incompatibility between the client and server. Possible values are:

>	Version, release, or level at the server is later than level at the client.
<	Version, release, or level at the client is later than level at the server. Manual intervention is required.
?	Server, client, or both are in an unknown state. No comparison is possible.

client state
: Indicates the current state of the program on the client. Possible values are:

a	Applied
c	Committed
u	Unknown (or error). Manual intervention is required. Program must be reinstalled.
r	Required program subset installation.

program name
: Indicates the name of the program that is to be installed or updated. This may be the name of a program subset.

version
: Indicates the current version of this program on the server.

release
: Indicates the current release of this program on the server.

level
: Indicates either the initial level of this program installed on the server (if this is an install record) or the current level (if this is an update record.)

program copy file name
: For an install record, indicates the name of the program copy file to use when installing this program. For an update record, this field contains dashes (−), and the program requires an update, not a complete installation.

client level
: Indicates the current update level of the program on the client at the time the update starts.

Special Mount Considerations

All of the required mounts for active code service are made **readonly**. In some cases, however, there may be files that cannot be removed from those directories and to which programs must have both read and write permission.

Following is a general procedure for making such files available with the proper permissions:

1. Before performing the remote mounts required for active code service, verify that the file or directory that must be available for read and write access is accessible through the **/native** path to the local filesystem. For more information see "Looking at Covered Files and Directories" on page 11-15 in this book.
2. Perform the required active code service mounts.
3. Mount the **/native** path of the required directory, with read and write permission, over the equivalent directory in the file structure mounted from the server.

To perform these mounts and unmounts automatically, add the appropriate **mount** commands to **/etc/rc.actvsrvc** and **umount** commands to **/etc/rc.unactvsrvc**.

For example, if a licensed program requires read and write permission to the directory **/usr/lpp/r**, you can add commands similar to the following to the **/etc/rc.actvsrvc** file before the active code service mounts are performed:

```
IF [ ! -d /native/usr/lpplr ]
THEN ECHO "/native/usr/lpplr is not available"
 EXIT
FI
```

After the lines that perform the required active code service mounts, add the necessary command for performing the local mounts:

```
mount /native/usr/lpplr
```

To unmount such directories or files automatically, include **umount** commands similar to the following in the **/etc/rc.unactvsrvc** file:

```
umount /usr/lpp/r
```

Providing Passive Code Service

In a passive code service environment, one or more servers make program copy files (including copies of program subsets and program updates) available to clients. Once the program copy files exist on the server, the client can mount the appropriate server directory and use the program copy files as input for **installp** or **updatep** (rather than using AIX diskettes, for example). In a passive code service environment, a client must install a complete program; since it actually runs in the stand-alone state, the passive code service client cannot make use of program subsets.

The **bffcreate** command creates program copy files. If you only want to create program copy files (that is, you do not want to install the program on the server), run **bffcreate** from the command line. If you want to create program copy files and install the program on the server at the same time, use the **installp -b** command; the **-b** flag causes **installp** to run **bffcreate** in addition to installing the program. The **bffcreate** command puts program copy files (for both complete programs and program subsets) in the **/usr/lpp.install** directory; program update files go into the **/usr/lpp.update** directory.

Note: The **bffcreate** command processes only files that are in backup format (not, for example, files that are in **tar** format). However, if files are in backup format, **bffcreate** can process them, even if **installp** cannot install them.

For more information about **bffcreate**, see **bffcreate** in *AIX Operating System Commands Reference.*

You can use passive code service to install or update any support programs except VRM and the kernel. (See Figure 11-6 on page 11-66 for information about the methods you can use for installing and updating all support programs.) Use the following procedure to install or update SNA Services, Distributed Services, VRM Baseband Adapter Device Driver or IBM Token-Ring Adapter Device Driver at a client:

1. Mount the appropriate server directory on the client, for example:

   ```
   mount -n server /usr/lpp.install /usr/lpp.install
   ```

   OR

   ```
   mount -n server /usr/lpp.update /usr/lpp.update
   ```

2. Run **installp** or **updatep**, using the **-d** flag to specify the path name of the program copy file:

   ```
   installp -d /usr/lpp.install/program.vv.rr
   ```

   OR

   ```
   updatep -ac -d /usr/lpp.update/updt.yyddd.nnn
   ```

Setting Up Automatic Remote Mounts

You may want to have some of your remote mounts happen automatically when the system starts up to provide access to the remote files whenever the network is operating. Other files or directories that you do not use frequently can be mounted manually using the **mount** command. You can use two different methods to mount remote file systems automatically:

1. Add a stanza to **/etc/filesystems** to describe the mount and then mount it from **/etc/rc.ds**,
2. Create a list of files and directories to be mounted, and use a shell script to perform the mounts.

"Using Distributed Services to Provide Code Service" on page 11-54 provides a description of how to set up the first method for a specific configuration in **/etc/filesystems**. "Creating a Single-System Image" on page 11-29 provides a description of how to set up the second method, including the shell scripts to help perform the mounts. If you are going to set up a single-system image across the network, set up that configuration before defining any additional mounts in **/etc/filesystems**. You may not need additional mounts. The following procedure describes a method useful to mount major file systems using stanzas in **/etc/filesystems**.

Note: You must have completed the configuration of a basic system (see "Configuring a Basic Distributed Services System" on page 11-17) before any automatic mounts will work.

Performing Automatic Mounts through /etc/filesystems

1. Use your favorite editor to edit **/etc/filesystems**.
2. Add a stanza to this file to describe each remote mount. Use the `type` attribute to define the set of automatically mounted file systems.
3. Save the file.
4. Edit **/etc/rc.ds**. Add a **mount** command to the end of the file. Use the **-t** flag to mount all of the automatically mounted file systems.
5. Save the file.

The next time the system starts up, the system will try to mount the specified remote file systems.

More Detailed Information

1. **Edit /etc/filesystems**

   The **/etc/filesystems** file describes file systems that the system can mount. In this file you can describe remote file systems that the system can mount as well as local file systems (such as **/dev/hd1**). Each defined file system has a stanza in this file that describes the characteristics of that file system. Stanzas are separated from each other by at least one blank line. Stanzas have the format:

   mount_point:
   　　attribute = *value*
   　　　.
   　　　.
   　　　.

   In this general format, *mount_point* is the place in your local file system where you want to mount the described file system, *attribute* is one of many defined attributes that can be listed in this file, and *value* is a value assigned to that attribute. Usually, a stanza contains more than one attribute = value pair. Refer to *AIX Operating System Technical Reference* for a complete description of the format for this file.

2. **Add a Stanza**

   Add a stanza for each remote mount that you want to perform automatically. Each stanza should contain at least the following information:

   mount_point:
   　　`nodename` = *remote nickname*
   　　`type` = *type string*
   　　`dev` = *path name*
   　　`mount = false`
   　　`inherit = true`

   In this general format, the parameters have the following meaning:

   remote nickname The nickname that you assigned to the remote system in **ndtable**.

   type string A unique string of characters that you define to represent a group of mounted file systems.

   path name The path name on the remote system that locates the file system to be mounted on that system.

   The entry, `mount = false` prevents the system from trying to mount this file system before Distributed Services is started, and also prevents the **mount all** command from trying to mount this file system.

   For example, the following stanza describes a file system to be mounted on the **/usr/man** directory. The file system is found on the remote system `zeus` in directory

/usr/man. We have chosen a type of `dsmount` to designate it as a file system that is mounted from **/etc/rc.ds**.

```
/usr/man:
        nodename = zeus
        type = dsmount
        dev = /usr/man
        mount = false
```

3. Save the file.
4. **Edit /etc/rc.ds**

   The file **/etc/rc.ds** contains shell commands that start SNA and Distributed Services. At the end of that file, add a line of the following form:

   `mount -t` *type string*

   In this format, *type string* is the same value that you assigned to the type attribute in **/etc/filesystems** for the file systems that you want mounted automatically. In the case of the previous example, add the following line at the end of **/etc/rc.ds**:

   ```
   mount -t dsmount
   ```

   This instruction tries to mount all file systems defined with a type of `dsmount`.
5. Save the file.

The next time that the system starts, it will try to mount the file systems that you specified in this procedure.

Looking at the Remote System's File Tree

Use the following procedure to set up a mount that allows you to look at the complete file tree on the remote system:

1. Make a directory in the root directory to receive the mount. Use the nickname of the remote system as the directory name. For example, to mount the file tree from zeus:

   ```
   mkdir /zeus
   ```

2. Add a stanza to **/etc/filesystems** to define the file system mount. For example, to mount the file tree from zeus:

   ```
   /zeus:
     nodename = zeus
     type = sysmounts
     dev = /
     mount = false,inherit
   ```

3. Add a line to the end of **/etc/rc.ds** but before any other remote mounts to perform the mount. For example, to mount the zeus file tree and perform other automatic mounts, the following lines might appear at the end of **/etc/rc.ds**:

```
mount -t sysmounts
mount -t dsmount
```

Building Distributed File Trees

The following are points to consider when planning and managing distributed file trees:

To Select the Type of Mounts to Use

1. Restrict your use of file-on-file mounts to special cases:
   - The files required to implement the Single-System Image Environment (see "Creating a Single-System Image" on page 11-29).
   - Read-only mounts of programs that you want to run remotely, but which reside in directories that you do not want to mount as a whole (perhaps to avoid the problems discussed in identifying and managing dependencies on page 11-87).
2. Use directory mounts to build those portions of the file tree to which users will have write access.

More Detailed Information

1. Restrict your use of file mounts to special cases:
   - The files required to implement the Single-System Image Environment (see "Creating a Single-System Image" on page 11-29).
   - Read-only mounts of programs that you want to run remotely, but which reside in directories that you do not want to mount as a whole (perhaps to avoid the problems described in the discussion of identifying and managing dependencies on page 11-87).
2. Use directory mounts to build those portions of the file tree to which users will have write access.

   To understand the problems that using file-on-file mounts to build file trees can cause, you need to remember the relationship between a file tree and a file system and to remember how the enhanced mount facility has changed that relationship.

   - The restrictions against linking or renaming files across file system boundaries still remain.
   - The new mount facility now makes it possible for two files in the same directory to be in two different file systems and for a file to be in a different file system than its parent directory.

When selecting what directories or files to mount from a program server, consider the following:

To Select Mounts from a Program Server

1. The following directories contain programs that cannot (or should not) be run from a client node:
   - **/bin**
   - **/etc**
   - **/lib**
2. The following directories, in general, contain programs or support facilities that can be accessed from a client node:
   - **/usr/bin**
   - **/usr/lpp**
   - **/usr/lib**
   - **/usr/include**
   - **/usr/mail**
   - **/usr/news**

More Detailed Information

1. The following directories contain programs that cannot (or should not) be run from a client node:

   /bin This directory contains the **sh**, **csh**, and **su** commands, as well as many other basic programs that you want to have available at all times.

   /etc This directory contains system administrative commands, such as **fsck**, **mkfs**, **shutdown**, **devices**, **minidisks**, **mount**, and **umount**. It also contains many system daemons, such as **cron** and **qdaemon**. Not only do these commands need to be available at times, but many can only be run locally. In addition, it contains configuration files that are specific to a particular system, such as **filesystems**, **system** and **master**.

   /lib This library contains the shared library (**sharsys.*xxxx***) used by many AIX commands. A local copy of this library needs to be available for use when the server is off line (see the discussion of identifying and managing dependencies on page 11-87).

2. The following directories, in general, contain programs or support facilities that can be accessed from a client node:

Directory	Possible Restrictions
/usr/bin	This directory can contain set-user-ID and set-group-ID programs that a remote user can run only if the supporting ID translation entries exist in the Network Users/Groups Table (see page 11-120). This directory can contain licensed program-installed programs that cannot (or should not) be run by remote users.
/usr/lpp	This directory contains locally installed licensed programs. Distributing licensed programs is subject to the licensing agreements, restrictions, and requirements imposed by their owners. See "Using Distributed Services to Provide Code Service" on page 11-54 for more information about distributing programs. **Note:** Licensed programs usually install some programs in other system directories, too, such as **/usr/bin**.
/usr/lib **/usr/include** **/usr/news**	
/usr/mail	To share this directory, the **mail** group (GID **6**) needs to be translated at each node.

In addition, when data files, system files, or programs reside on another node, each client node needs to identify critical dependencies and have a plan for managing those dependencies if those files or programs should be unavailable. These problems usually fall into one of two areas:

The server node is not online when the client node starts up.

To manage this problem:

- Be sure that there are local copies of any programs that users must have access to at all times.
- Alternatively, you might customize a secondary program server. Files and directories from this system would not be mounted automatically, but would be available if the first server were inaccessible.
- Use a shell program to issue automatic remote mounts, so that they can be retried until successful. See "Automatic Mount Procedures" on page 11-89 for a complete discussion of this alternative.

The server node goes off line while the client node is using remote resources.

Depending on what system resources you are using at the server, the following are possible problems you can have under these circumstances:

- Users can no longer log in or run commands that read the **/etc/passwd** file.

  For example, a user tries to log in with a valid user name, but receives the following message:

  ```
  You entered a login name or password that is not valid.
  ```

  or a user tries to use the **su** command to change to a valid user ID, but receives the following message:

  ```
  Unknown id
  ```

  The AIX Operating System must read the **/etc/passwd** file to get the UID that corresponds to a user name. If the **/etc/passwd** file is remote, the local AIX Operating System cannot access it when the remote system is off line. This problem is complicated by the fact that, under these circumstances, if a user with superuser (system group) authority is not logged on when the server goes off line, it will not be possible to unmount remote files or directories or to shut down the client node until the server is accessible again. In addition, if no user is logged on, no one will be able to log on until the server is accessible again.

  To solve this problem, you must provide some way for a nonsystem group user to unmount an inaccessible **/etc/passwd** file or for the system to monitor that file and to unmount it when it becomes inaccessible. To allow a non-system group user to unmount an inaccessible password file, compile the sample program **umpasswd.c** contained in the samples directory, **/usr/lpp/ds/samples**. Use the following commands to make this program available to users:

  ```
  # cd /usr/lpp/ds/samples
  # cc umpasswd.c -o umpasswd
  # chown root umpasswd ; chgrp system umpasswd
  # chmod u+s umpasswd
  # cp umpasswd /etc
  ```

  The `chmod u+s` command makes this a set-user-ID program (the **cc** command has already made it executable by all users). This gives a user the necessary authority to issue the **uvmount** system call in the program. (Note that the program only unmounts the password file if it cannot open it. Otherwise, it exits without taking any action.)

  To have the system run this program periodically, schedule it to be run by the **cron** command. Then even if no user is logged on, the password file will eventually be unmounted once it becomes inaccessible.

Automatic Mount Procedures

There are a number of automatic mount procedures, depending, in part, on whether or not remote mounts are inherited.

Mounts That Are Not Inherited

If your remote mounts are not inherited, automatic mount procedures need only be implemented at the node issuing the remote mount.

There are at least two ways to reschedule mounts:

- Have a shell program use the **at** command to rerun itself.
- Run a background shell program that does not end until its mounts are all successful or until it reaches some specified limit.

Rescheduling Mounts: the at Command

The sample shell program **retry_mount** in the samples directory **/usr/lpp/ds/samples** illustrates one way to use the **at** command to reschedule mounts. This shell program issues a mount, tests its return value, and reschedules itself if the mount is unsuccessful. This process continues until the mount succeeds.

Run **retry_mount** from the **/etc/rc.ds** initialization file if any remote mounts fail. For example, see the following command list:

```
mount -t remote >/usr/adm/mnt.msg 2>&1 || retry_mount remote 5
```

The commands in this list are separated by two `||` (vertical bars). This causes the second program (`retry_mount`) to run only if the first program (`mount`) fails. **retry_mount** also uses ***conditional substitution*** to assign a value to the user-defined variable `msgpath`:

```
msgpath=${msgpath=/usr/adm/mnt.msg}
```

This line assigns the value `/usr/adm/mnt.msg` to the variable `msgpath` if that variable has not already been set on the command line. Conditional substitution can be useful if you expect to use a particular value, but want to be able to change that value if necessary. To reset this variable, enter a command such as the following:

```
msgpath=/dev/null retry_mount remote 5
```

This command line assigns all output from the mount commands to **/dev/null**.

See the **sh** command in *AIX Operating System Commands Reference* for a description of available command-list separators and terminators and for a discussion of user-defined variables and conditional substitution.

Using a Shell Program that Sleeps

Another automatic mount procedure runs a background shell program that sleeps for a specified interval and then reissues the required mounts until they are all successful. The **do_mounts** program in the **/usr/lpp/ds/samples** directory is an example of such a program. The only exit from this program appears in the command list:

```
mount -t $1 >>${msgpath} 2>&1 && exit
```

The `&&` separator causes `exit` to run only if the mount is successful.

On the other hand, you may want to limit the number of times your system retries a mount. You may not want to tie up system resources indefinitely. Perhaps some nodes provide only limited file services or are not always on line. In this case, you could use a shell program such as **domounts** in **/usr/lpp/ds/samples** to limit mount attempts. The **domounts** shell program accepts an optional third command line argument that specifies the number of times that the program is to attempt the mount. If you do not specify a third argument, the program continues to loop until the mount is successful. The following line sets the default limit (`0`):

```
limit=${3-0}
```

The program, using conditional substitution to assign a value to a variable named `limit`, checks to see if a third argument is set on the command line (`$3`). If it is, `domounts` assigns that value to `limit`. If it is not set, `domounts` assigns the value `0` to `limit`.

Have the **rc.ds** file run `do_mounts` ***in the background*** so that **rc.ds** can complete system startup. Use lines in **rc.ds** such as:

```
echo Mounting remote directories ...
domounts primary 2 &
domounts secondary 5 10 &
```

Inherited Mounts

To get all the remote mounts in an inherited file tree, the node issuing an inherited mount must not issue that mount until after the tree has been built at the Server. Following are two solutions to this problem, one implemented from the server node (the “Smart Server”), and one implemented from the client node (the “Smart Client”).

Smart Server

To implement this solution, a server node uses the **dsstate** command to block incoming requests until the file tree to be inherited has been completed or until a specified interval has passed. Then it re-enables incoming server requests and, if necessary, continues re-issuing the mounts. See **/usr/lpp/ds/samples/repeatmounts**.

By blocking incoming requests, the server node prevents other nodes from mounting a partially finished file tree. This solution is suited to environments where there is only one server node building file trees to inherit (and thus blocking incoming requests).

To enable this approach, the server node includes the following lines in **/etc/rc.ds**:

```
dsstate -p100 -k -ce -se -sb
dsstate -aa
repeatmounts filetree 1 10 4 &
do_mounts remote 5 &
```

Each client includes the following lines in **/etc/rc.ds**:

```
dsstate -p100 -k -ce -se -sa -aa
domounts primary 2 &
domounts secondary 5 10 &
```

Note: Be sure that the **/etc/filesystems** file stanzas of all inherited mounts contain the attribute **mount=inherited**.

Smart Client

To implement this solution, the server node keeps a log of the **type** attribute of each successful mount. This can be done with the sample shell program **recordmounts** in **/usr/lpp/ds/samples**.

After each successful mount, **recordmounts** writes the mount type in a file. The client node checks this log file, and does not attempt an inherited mount until this file contains a specified mount type, as illustrated by **clientmounts** in **/usr/lpp/ds/samples**. This shell program uses its process ID ($$) to create a temporary directory and null file on which to mount the server's log file. It keeps scanning this file until it contains a line matching the mount `type` that **clientmounts** is looking for. Only then does it issue its inherited mounts.

The shell program **buildtrees** in **/usr/lpp/ds/samples** manages the building of inherited file trees at the server. It ensures that server requests are not re-enabled until all the specified file trees have been completed. The **rc.ds** file at the server node contains the following instructions:

```
dsstate -p100 -k -ce -se -sb
dsstate -aa
buildtrees filetree1 filetree2 &
```

The **rc.ds** file at the client node contains the following instructions:

```
dsstate -p100 -k -ce -se -sa -aa
clientmounts node_a filetree1 2 /u &
clientmounts node_a filetree2 5 /library &
domounts remote 5 10 &
```

Using Distributed Services Menus

Note: You must have superuser authority or be a member of the system group to create or modify Distributed Services tables. Any user can browse the tables for information.

To use a windows interface to build, examine, or modify the Distributed Network Node Table, the Network Node Security Table, the Network Users/Groups Table, or the Distributed IPC Queues Table, run one of the following commands:

ndtable To access the Distributed Network Node Table (see "Working with the Distributed Network Node Table" on page 11-96)

ugtable To access the Network Users/Groups Table (see "Working with the Network Users/Groups Table" on page 11-107)

ipctable To access the Distributed IPC Queues Table (see "Working with the Distributed IPC Queues Table" on page 11-124)

pwtable To access the Network Node Security Table (see "Working with the Network Node Security Table" on page 11-104)

In addition, the **dsldxprof** command allows you to modify the Network Users/Groups Table with information stored in an ASCII file. See "Using the dsldxprof Command" on page 11-121 for more information.

These commands use data files that reside, by default, in the local directory **/etc/profsvcs/pfslocal**. When these commands display the selected table, the table appears in a window similar to that shown in Figure 11-7 on page 11-94 for the Distributed Network Node Table.

The ***command bar*** above the window shows the commands that you can use at the current time. If you select an entry in the table, the available commands in the command bar change. In addition, ***pop-up panels*** appear when you make certain selections. Figure 11-8 on page 11-94 lists the commands and their functions.

Depending on the display station, either the > or » symbol appears before certain items on the display screen. Each item (word, phrase, or symbol) that follows this symbol is called a ***button*** and you can select it as a unit from the display screen. If an area filled with periods (...) follows this symbol, this area is called an ***input field***. An input field allows you to type in or change the data required in an entry.

You can use a mouse to select entries, buttons, and input fields, or you can use Usability Services Functions from the keyboard. To identify the necessary keys or key sequences, see the *Usability Services Functions* keyboard template that corresponds to your keyboard. The functions you will probably use most frequently are **Command Bar** (to move between the display window and the command bar), **Do**, **Help**, **Quit**, **Select**, and **Tab**.

```
>>PICK  >>UPDATE  >>SORT  >>ADD  >>PRINT                    >>CLOSE

  Distributed Network Node Table:  Node 2000002A

Last UPDATE at 14:25                 Network Name:  IBMRTDS

Remote      Remote      Node        Data Link   Connection   Attachment
Nickname    Node ID     Security    Type        Profile      Profile

>>D57       2000068E    Secure      Ethernet    2000068E     2000068E
>>D92       2120B356    Secure      Ethernet    2120B68E     2120B68E
>>Admin     384D6512    Secure      Ethernet    384D6512     384D6512
```

A5ACG017

Figure 11-7. Sample Table Window

Command	Used In	Function
add	All	Adds an entry to the active table. A pop-up panel appears to allow you to enter information to define the new entry.
change	All	When an entry in the active table is selected, this button allows you to change the information in that entry. A pop-up panel appears to allow you to change the information. Not all information can be changed.
close	All	Closes the active table and returns you to the operating system.

Figure 11-8 (Part 1 of 2). Distributed Services Table Commands

Command	Used In	Function
copy	All but **ipctable**	When an entry in the active table is selected, this button allows you to copy that entry to create a new entry in the table. A pop-up panel appears to allow you to specify the name of the new entry and to change any of the information.
delete	All	When an entry in the active table is selected, this button allows you to delete the selected entry. A pop-up panel appears to allow you to confirm or cancel the delete operation.
pick	All	This button allows you to select which entries to display in the window by specifying a pattern, a specific name, or an attribute to match. A pop-up panel appears to allow you to enter pattern matching strings to select groups of entries to display. The panel allows you to select from any of the major fields displayed in the active table.
print	All	This button allows you to write the list displayed on the screen to a file or to the printer. A pop-up panel appears to allow you to specify options. If you select printing to a file, a second pop-up panel appears for specifying the file name and whether to replace an existing file of the same name.
sort	All	This button allows you to select the order in which to list the profiles that appear in the active table. A pop-up panel appears to allow you to select a field on which to sort, and to specify the order of the sort. You can select one field to sort by and one sort order.
update	All	This button allows you to refresh information displayed in the active table to show changes made from other windows. The system does not automatically update the display. The time of the last `UPDATE` command is displayed so that you know when you last updated the list. For example: `Last Update at 15:30`

Figure 11-8 (Part 2 of 2). Distributed Services Table Commands

Working with the Distributed Network Node Table

The ***Distributed Network Node Table*** defines each remote node that is known to the local node. Each entry in the table identifies the remote node's nickname (optional), node ID (NID), node security, data link type, and connection profile name. This information is stored in SNA profiles. Distributed Services can create and work with two types of node tables:

Static Static node tables are made up of permanently constructed SNA profiles that always exist in the profile data base as long as the respective nodes are defined to Distributed Services. You can modify these profiles as needed using the SNA profile management command, **snaconfig**.

Dynamic Dynamic node tables are made up of information to build the needed SNA profiles. The information is contained in the following files in **/etc/profsvcs/pfslocal**:

pfsndtab	Contains all table information except the BIND password.
.pfsndtab	Contains index information to **pfsndtab**.
pfsndpw	Contains table information for the BIND password.
.pfsndpw	Contains index information to **pfsndpw**.

The system builds each profile as it is needed, and discards the profile when it is not being used. Dynamic node tables allow systems to mount node table information regarding system nicknames and node IDs from a node table server, while still maintaining separate BIND password tables for each system (see "Creating a Node Table Server" on page 11-25)

You can define a network equally well using either or both types of node tables. However, if you define the same node in both tables only the information in the static table for that node is used. Use the dynamic node table to create a node table server function (see "Creating a Node Table Server" on page 11-25).

Use the **ndtable** command to work with node tables.

Starting the ndtable command

1. Enter the **ndtable** command on the command line:

   ```
   # ndtable
   ```

   A pop-up panel appears requesting you to identify the node.

   ```
   Enter Node ID Nickname       ».............
   (Default is the local id.)
   ```

   If you specify a NID or node nickname, the files found in **/etc/profsvcs/pfslocal** on the specified node are accessed. If you specify a directory name, the command looks under **/etc/profsvcs** for a directory with that name and uses the files found in **etc/profsvcs/pfslocal** under that directory. The default value is the local node ID. When processing the table for the local node, the command uses the files in **/etc/profsvcs/pfslocal**.

2. Press **Do** when you have entered the correct value. A pop-up panel appears asking for the type of node table to access.

   ```
   Type of the Node Table    » Static    » Dynamic
   ```

3. Select the desired type of table and press **Do**. The node table appears. If you select *static*, only those nodes that have static profiles defined appear in the table. If you select *dynamic*, only those nodes that have dynamic profiles defined appear in the table.

4. Once the table is displayed, you can add, delete or change the information in that table.

5. Select **Close** on the command bar to end the session.

Defining a Static Node

Selecting ADD allows you to add a new entry to the list.

When you select ADD from the command bar of the Distributed Network Node Table window, you see the following pop-up panel:

```
Remote Nickname          ».............

Remote Node ID           »........

Node Security            »None
                         »Secure

Data Link Type           »Ethernet
                         »SDLC
                         »Token Ring
                         »802.3
```

You must provide a Remote Node ID. All other input fields are either optional (Remote Nickname) or have default values (Node Security, Data Link Type).

The Remote Nickname can contain numbers and both uppercase and lowercase letters. In the Node ID input field, **ndtable** converts lowercase alphabetic characters to uppercase.

Note: If your system has a communication password set, the system asks you to enter that password the first time that you try to add, delete or change a BIND password in any **ndtable** session.

If you select Secure, you are given the following additional choices:

```
PLEASE SELECT BIND PASSWORD TYPE.
  »30-80 CHARACTER PHRASE PASSWORD

  »16 CHARACTER HEX BIND PASSWORD
```

When you select one of the password types, another panel appears to allow you to enter the password twice.

When you select one of the data link types from the **ADD** panel, another panel appears asking you to enter the data link device name. The name you select depends on which physical device you use to make the connection to the node. Figure 11-1 on page 11-7 lists the physical connections and the choices for each.

When you have made all your choices in the **ADD** pop-up, press **Do** to add the new node to the list.

Defining a Dynamic Node

Selecting ADD allows you to add a new entry to the list.

When you select ADD from the command bar of the Distributed Network Node Table window, you see the following pop-up panel:

```
Remote Nickname                »..............

Remote Node ID                 »........

Node Security                  »None
                               »Secure

Prototype Connection Profile   »..............

Prototype Physical Link
  Profile                      »EDEFAULT......

Command/Selection Sequence     »....................
```

You must provide both a Remote Nickname and a Remote Node ID. All other input fields have default values. The Remote Nickname can contain numbers and both uppercase and lowercase letters. In the Node ID input field, **ndtable** converts lowercase alphabetic characters to uppercase.

Note: If your system has a communication password set, the system asks you to enter that password the first time that you try to add, delete or change a BIND password in any **ndtable** session.

If you select Secure, you are given the following additional choices:

```
PLEASE SELECT BIND PASSWORD TYPE.
  »30-80 CHARACTER PHRASE PASSWORD

  »16 CHARACTER HEX BIND PASSWORD
```

When you select one of the password types, another panel appears to allow you to enter the password twice.

Use the default prototype connection profile for all data link types. You may change the default prototype physical link profile name to match the network adapter being used (see Figure 11-1 on page 11-7). You can also using **snaconfig** if the default profiles do not meet your needs (see *SNA Services Guide and Reference*).

The command/selection sequence is used for X.21 selection sequence or a Smart Modem command sequence only. Leave it blank for other connections.

X.21 Selection Sequence

The selection sequence is a series of up to 255 ASCII characters that determine the destination of the information on the network. Use only the following characters: digits 0 to 9, *, +, , (comma), -, . (period), and /. SNA Services adds a + to the end of the sequence as a terminator.

Smart Modem Command Sequence

The modem command sequence is a series of up to 79 ASCII characters that control the modem's call establishment and data transfer options. This sequence is transferred to the modem as a single character string using asynchronous communications when the attachment is first started. A terminating carriage return is automatically added to the end of the sequence. Once the modem's data set ready (DSR) circuit becomes active, the modem switches to synchronous mode for the remainder of the call.

Changing An Existing Static Node Entry

When you select CHANGE, a pop-up panel appears containing the fields that you can change. When you change an existing node entry in the local node, the following actions occur, depending on the changes you have made:

- If you change the node nickname or node ID, all tables—the Distributed Network Node Table, the Network Users/Groups Table, and the Distributed IPC Queues Table—are updated to reflect that change.
- If you change node security the first time in a session, you are prompted for a communications password if one is defined. The BIND password is encrypted with the Communication Authority Password key and stored in the SNA Services connection profile. (For more information see *SNA Services Guide and Reference.*) The BIND password must be the same on both communicating nodes (that is, in the connection profile on each node).

  If you change node security from Secure to None, the node password is deleted from the SNA Services connection profile.

Changing An Existing Dynamic Node Entry

When you select CHANGE, a pop-up panel appears containing the fields that you can change. When you change an existing node entry in the local node, the following actions occur, depending on the changes you have made:

- If you change the remote nickname or remote ID, no changes are made to the Network Users/Groups Table or the Distributed IPC Queues Table.
- If you change the node security the first time in a session, you will be asked for the communications password if one is defined.
- If you change node security to secure, you will be asked for a BIND password for that connection. It must be the same as the BIND password used to define the other end of the connection on the remote system. This password is encrypted and stored in **pfsndpw**.
- If you change the node security to none, the BIND password for that connection is deleted from **pfsndpw**.
- If you change the prototype profiles or the command/selection sequence fields, the changes are reflected in **pfsndtab**.

Deleting An Existing Remote Node Entry

Warning: The `DELETE` command permanently erases the selected profile entry.

When you delete an existing *static* entry from the local node's database, **ndtable** does the following:

- Deletes the node's SNA Services Protocol profiles, if they are not used by any other profiles.
- Removes any keys in the IPC key profile that contain the node ID or nickname of the deleted node.
- Removes all Network Users/Groups Table entries that identify the deleted node as the originating node.

When you delete an existing *dynamic* entry from the local node's database, **ndtable** deletes the entry from **pfsndtab** and **pfsndpw** (if the connection is BIND password protected). The Network Users/Groups Table and Distributed IPC Queues Table are not affected.

Working with the Network Node Security Table

The ***Network Node Security Table*** defines the BIND passwords for all secure nodes defined in a *remotely mounted* dynamic node table. See "Creating a Node Table Server" on page 11-25 for information about setting up a remotely mounted dynamic node table. Use **ndtable** (see "Working with the Distributed Network Node Table" on page 11-96) to define and change BIND passwords for static node tables and *local* dynamic node tables.

Use the **pwtable** command to access the Network Node Security Table.

Starting the pwtable command

1. Enter the **pwtable** command on the command line:

   ```
   # pwtable
   ```

   A pop-up panel appears requesting you to identify the node.

   ```
   Enter Node ID Nickname        ».............
   (Default is the local id.)
   ```

   If you specify a NID or node nickname, the files found in **/etc/profsvcs/pfslocal** on the specified node are accessed. If you specify a directory name, the command looks under **/etc/profsvcs** for a directory with that name and uses the files found in **etc/profsvcs/pfslocal** under that directory. The default value is the local node ID. Profiles for the local node are in **/etc/profsvcs/pfslocal**.

2. Press **Do** when you have entered the correct value. The Network Node Security Table appears.

3. Once the table is displayed, you can add, delete or change the information in that table.

4. Select **Close** on the command bar to end the session.

Adding a BIND Password

When you select ADD from the command bar, a pop-up panel appears for you to enter the nickname of the node for which you are adding a password.

1. Enter a nickname that is already defined in a mounted dynamic node table and press **Do**. A pop-up panel appears:

```
PLEASE SELECT BIND PASSWORD TYPE.
  »30-80 CHARACTER PHRASE PASSWORD

  »16 CHARACTER HEX BIND PASSWORD
```

2. Select one of the password types. Another panel appears to allow you to enter the password twice.
3. Enter the password as requested and press **Do**. The password is added to **pfsndpw** and the Network Node Security Table appears.
4. Select **Close** to exit.

Changing a BIND Password

Selecting CHANGE allows you to change the BIND password for a connection that you have selected from the table.

When you select CHANGE from the command bar, a pop-up panel appears that displays the nickname of the selected connection. You can change the nickname by typing in a new name, but if you do you must also change the BIND password.

Press **Do**. Another panel appears:

```
PLEASE SELECT BIND PASSWORD TYPE.
  »30-80 CHARACTER PHRASE PASSWORD

  »16 CHARACTER HEX BIND PASSWORD
```

1. Select one of the password types. Another panel appears to allow you to enter the password twice.
2. Enter the password as requested and press **Do**. The password is added to **pfsndpw** and the Network Node Security Table appears.
3. Select **Close** to exit.

Deleting a BIND Password

Selecting DELETE allows you to remove the BIND password for a connection that you have selected from the table.

When you select DELETE from the command bar, a pop-up panel appears that displays the nickname of the selected connection. Press **Do** to delete the password. The entry in **pfsndpw** is deleted, and the Network Node Security Table is displayed again.

Working with the Network Users/Groups Table

The ***Network Users/Groups Table*** defines the users and groups from the local system that can access the distributed services functions. Access to these functions is in two categories:

Outbound This type of access defines what a local user (or group) can do when trying to perform operations with files and directories on other systems.

Inbound This type of access defines what users (or groups) on other systems can do when trying to perform operations with files and directories on the local system.

Access permissions are controlled by IDs that you assign to each user on your system and to each user from another system that accesses your system. The following terms describe the different forms that IDs can have in a distributed AIX environment:

local ID A numeric ID associated with a user or group name defined in the local **/etc/passwd** or **/etc/group** file. This ID is found in local i-nodes and is associated with locally running processes.

remote ID A numeric ID associated with a user or group name defined in a remote **/etc/passwd** or **/etc/group** file. This ID is found in remote i-nodes and is associated with a process running at the remote node.

network ID A numeric ID associated with a communication request; the ID that is passed between nodes on the network. This ID must be unique for each user on the network. Numbers such as employee ID or social security number work well as network IDs.

The same user name may correspond to different IDs on each node in a Distributed Services network. To satisfy the requirements of AIX permission checking, Distributed Services can perform one or both of the following ID translations to associate the IDs on one node with those on another node:

Translation	Function
Inbound	Translates the network IDs of incoming requests into their equivalent local IDs.
Outbound	Translates the IDs of outgoing requests into their equivalent network IDs.

Refer to "Distributed Services ID Translation" on page 11-112 for more information about how ID translation occurs.

Use the **ugtable** command to work with Network Users/Groups Tables.

Starting the ugtable command

1. Enter the **ugtable** command on the command line:

   ```
   # ugtable
   ```

   A pop-up panel appears requesting you to identify the node.

   ```
   Enter Node ID Nickname        ».............
   (Default is the local id.)
   ```

   If you specify a NID or node nickname, the files found in **/etc/profsvcs/pfslocal** on the specified node are accessed. If you specify a directory name, the command looks under **/etc/profsvcs** for a directory with that name and uses the files found in **etc/profsvcs/pfslocal** under that directory. The default value is the local node ID.

2. Press **Do** when you have entered the correct value. The Network Users/Groups Table appears.

3. Once the table is displayed, you can add, delete or change the information in that table.

4. Select **Close** on the command bar to end the session.

Adding a User or Group to the Network

Selecting ADD allows you to add a new entry to the list.

When you select ADD from the command bar of the Network Users/Groups Table window, you will see the following pop-up panel:

```
User/Group Entry                     »User  »Group

User/Group Local ID                  ».....
(opt. if U/G name entered exists)

User/Group Name                      »........
(Ignored if U/G Id entered)

Outbound Network ID                  »..........
Inbound Network ID                   »..........

Originating Node ID/Nickname         »..............
```

Select User or Group to indicate whether the entry describes a user (the default) or a group ID. If you are adding an entry to a local table, you must specify either a numeric UID (GID) or the user (group) name. If you specify a numeric ID, the corresponding name from the **/etc/passwd** or **/etc/group** file will be displayed (even if you specify a different name). If you are adding an entry to a remote table, you must specify a numeric ID, and no corresponding name will be displayed.

If you enter a NID in the Originating Node ID/Nickname input field, you must enter all alphabetic characters as uppercase characters. Unlike **ndtable**, **ugtable** cannot convert lowercase characters to uppercase, since you can enter a node nickname in the same field (a name that can contain numbers and both uppercase and lowercase characters).

You can have more than one entry in the table for a single local ID, but each entry must have the same outbound ID. If you try to add another entry that has a different outbound ID from the one already associated with that local ID, you will be asked if you want to change all the existing outbound IDs. You may then press **Do** to change all the outbound IDs or **Quit** to return to the **ADD** pop-up panel where you can change the information for the new ID.

Adding Many Users or Groups to the Network

When you must add a large number of users or groups to the Network Users/Groups Table, use the **dsldxprof** command from the operating system command line. This command loads an ASCII file of names into the Network Users/Groups Table.

Loading the Network Users/Groups Table

1. Use your favorite editor to create an ASCII file that defines the IDs that you want to add (see "Using the dsldxprof Command" on page 11-121 for the format of that file).
2. Load the contents of that file into the Network Users/Groups Table using the **dsldxprof** command. For example, if you name the ASCII file `newids`, then the following command loads the contents of that file into the Network Users/Groups Table:

   ```
   # dsldxprof -f newids
   ```

Changing Network Users/Groups Table Entries

Selecting `CHANGE` allows you to modify the description of an existing entry.

The change pop-up panel displays the values in the user/group entry selected. The name comment field will not be displayed if a remotely mounted node is being configured. The change may be either to an ID or a name and the same validation checks will be made for the name/ID as in adding and copying. If the outbound ID is changed and if other entries have the same local ID, then you see a pop-up panel that allows you to change the outbound IDs of all old entries to the new ID or to return to the add/copy pop-up panel to correct the outbound ID on the new entry.

Note: The system does not update the **passwd** or **group** files. You must run the **users** command to update these files if necessary.

Changing Many Network Users/Groups Table Entries

Use the **dsldxprof** command from the operating system command line to make many changes to the Network Users/Groups Table.

Changing Many Network Users/Groups Table Entries

1. Use **ugtable** to access the Network Users/Groups Table. Select PRINT to copy the Network Users/Groups Table without headers to a file.
2. Edit the file with your favorite editor.
3. For each entry that you want to change:
   a. Copy that line directly below itself.
   b. Insert ## characters at the start of the original copy of the line (this instructs **dsldxprof** to delete that entry).
   c. Change the second copy of the line to reflect the new information.
4. Save the file and exit the editor.
5. Use **dsldxprof** to make the changes:

```
# dsldxprof -f rmtable
```

Replacing All Network Users/Groups Table Entries

1. Use your favorite editor to create an ASCII file that defines the IDs that you want to add (see "Using the dsldxprof Command" on page 11-121 for the format of that file).
2. Delete the current entries in the Network Users/Groups Table and load the contents of that file into the Network Users/Groups Table using the **dsldxprof** command. For example, if you name the ASCII file `newids`, use the following command:

```
# dsldxprof -d -f newids
```

Deleting a User or Group

Warning: The DELETE command permanently erases the selected profile entry.

Selecting DELETE allows you to delete the entry that you have selected. If you delete an entry in the local Network Users/Groups Table, the kernel rebuilds its Network Users/Groups Table when you finish configuring the table and select CLOSE.

Note: The system does not delete any entries in the **passwd** or **group** files. You must run the **users** command to make any required updates to these files.

Deleting Many Users or Groups

Use the **dsldxprof** command to remove a large number of users or groups from the Network Users/Groups Table. To instruct **dsldxprof** to delete an entry, append ## characters to the first field of the line in the input file that describes the user or group to be deleted:

```
##-   U   200   200   200   *
```

To delete all entries in a Network Users/Groups Table, use the **-d** flag:

```
# dsldxprof -d
```

This command flag deletes all entries in the local Network Users/Groups Table. Use the **-n** flag to specify a remote Network Users/Groups Table (see "Using the dsldxprof Command" on page 11-121).

Distributed Services ID Translation

The AIX Operating System uses a combination of identification numbers (IDs) and permissions to provide data security. When a user logs in, the **/etc/passwd** file associates a user ID (UID) with the login name. Similarly, the **/etc/group** file associates one or more group IDs (GIDs) with the login name. When a user starts a process (runs a command), AIX associates the user's UID and GID(s) with the process. AIX also associates a UID and GID with each file or directory. Unless they are explicitly changed, the UID and GID associated with a file are those of the user who created it.

AIX also associates three permissions (read, write, and search or execute) for three classes of users (owner, group, and other) with each file or directory. A file's i-node contains information about its permissions, UID, and GID. A process can use a file only if the appropriate correlation between IDs (UID and GID) and permissions exists. (For more

information about file permissions, see "Protecting Files and Directories" in *Using the AIX Operating System*. For more information about i-nodes, see "I-nodes" on page 1-8.)

ID translation is the process of converting between a local ID or remote ID and a network ID. In respect to the local node, ID translation occurs in two directions:

forward translation
: The translation of IDs associated with running processes. A local ID is transmitted to a remote node. The remote node translates the ID to determine whether the process has access to the remote node and to do AIX permission checking.

backward translation
: The translation of IDs found in i-nodes. The local node translates a remote ID into its local equivalent to determine which local IDs own a remote file or directory.

At least two IDs (UID and GID) are associated with each process. Distributed Services translates IDs according to information in tables on each node. After a remote ID is translated to its local equivalent, AIX checks permissions by referring to the local **/etc/passwd** and **/etc/group** files. Thus, data security in a distributed AIX environment is comparable to that of an AIX system without Distributed Services.

Distributed Services loads profiles that contain the translation data into the AIX kernel. "Using Distributed Services Menus" on page 11-93 explains how to create and modify the profiles.

Inbound Translation

Each request from a client contains node ID (NID), UID, and GID information, and requires permission checking at the server. When a server receives a request, it uses the ID information of the request to select a row from the translation table. The row provides the equivalent local ID that AIX uses to perform the permission checking.

Note: The examples in this section demonstrate UID translation. Distributed Services translates GIDs in the same way.

Figure 11-9 on page 11-114 represents inbound translation. When the client node (NID `108133EB`, nickname `D94`) places a request on the network, the local ID and network ID of the requesting process are the same (`201`). When the server node (NID `208131AA`, nickname `D87`) receives the request, it translates the network ID into its local equivalent (`301`). To determine whether to give the process access to a file, the server node performs normal AIX permission checking; in this case, the file's owner (UID `301`) has all three permissions for the file (`-rwxr--r--`).

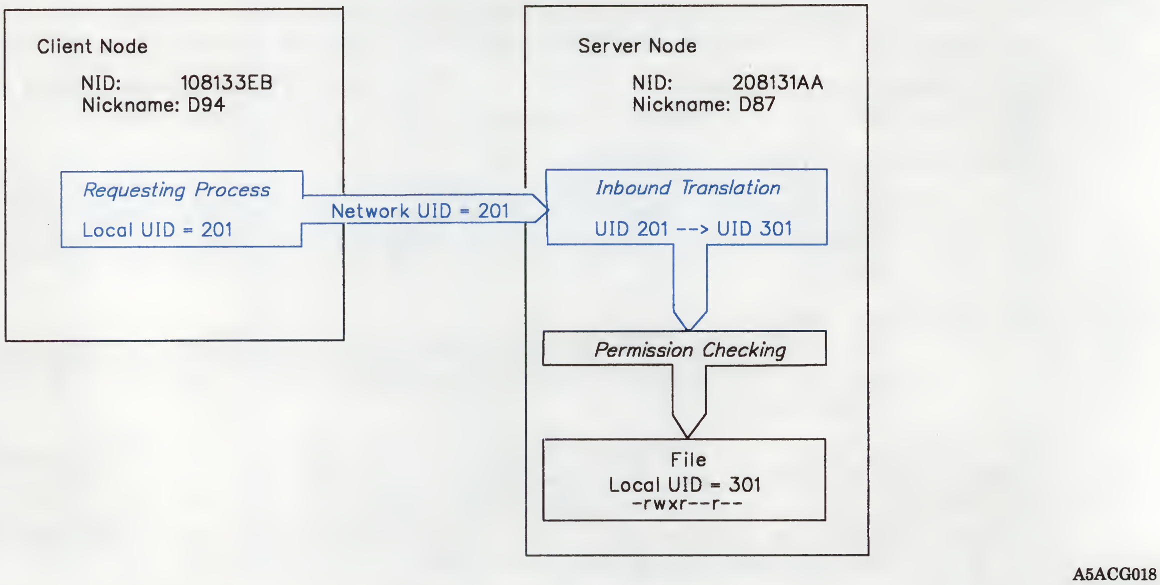


Figure 11-9. Inbound ID Translation

To perform the translation shown in Figure 11-9, the translation table on D87 contains the following data:

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
tom	U	301	201	201	D94

Node D87 translates the inbound network ID 201 into the local UID 301. The local UID 301 corresponds to the local user name tom.

In some cases, it is convenient to use wildcard entries in translation tables. An asterisk (*) in a field indicates that a specific ID is not required. In the following table, for example, a request with the inbound network ID 210 translates to the local UID 208, regardless of which node the request comes from:

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
tom	U	301	201	201	D94
don	U	208	210	210	*

The following example shows a wildcard character in the NID and inbound network ID fields of the third line. Any incoming request that does not match either of the first two lines in the table is mapped to the local `guest` ID.

```
Usr/Grp              Local          Network ID          Originating Node
 Name      U/G        ID        Outbound      Inbound    Name/Nickname

 tom        U         301          201          201      D94
 don        U         208          210          210      *
 guest      U         200          200            *      *
```

If a line in the translation table explicitly matches the UID and NID of an incoming request (that is, there are no wildcards), Distributed Services always performs that translation. However, with wildcards, an incoming request can match more than one translation line. According to the following table, a request with the inbound UID `210` and the NID `D94` matches three different local IDs (`208`, `212`, and `200`):

```
Usr/Grp              Local          Network ID          Originating Node
 Name      U/G        ID        Outbound      Inbound    Name/Nickname

 tom        U         301          201          201      D94
 don        U         208          210          210      *
 jean       U         212          212            *      D94
 guest      U         200          200            *      *
```

In such cases, since an inbound network ID can be translated to only one local ID, Distributed Services selects one of the possible translations according the the following priorities:

Priority	**Inbound UID**	**Inbound NID**
1	*explicit*	*
2	*	*explicit*
3	*	*

If no match occurs, Distributed Services rejects the request.

Note: Wildcards simplify ID translation by making it less restrictive. Consider the security requirements of your network when you decide how extensively to use ID translation wildcards.

Outbound and Inbound Translation

The principle of outbound translation is the same as that of inbound translation—one ID is translated into another according to data contained in a table. Outbound translation, however, occurs at the client node when the client places a request on the network. Outbound translation produces a network ID which can be different from both the local ID and the remote ID.

Figure 11-10 represents inbound and outbound translation working together. Before placing a request on the network, node D94 translates the local ID of the process (201) to produce the network ID (210). Upon receiving the request from the network, node D87 performs inbound translation and permission checking as usual.

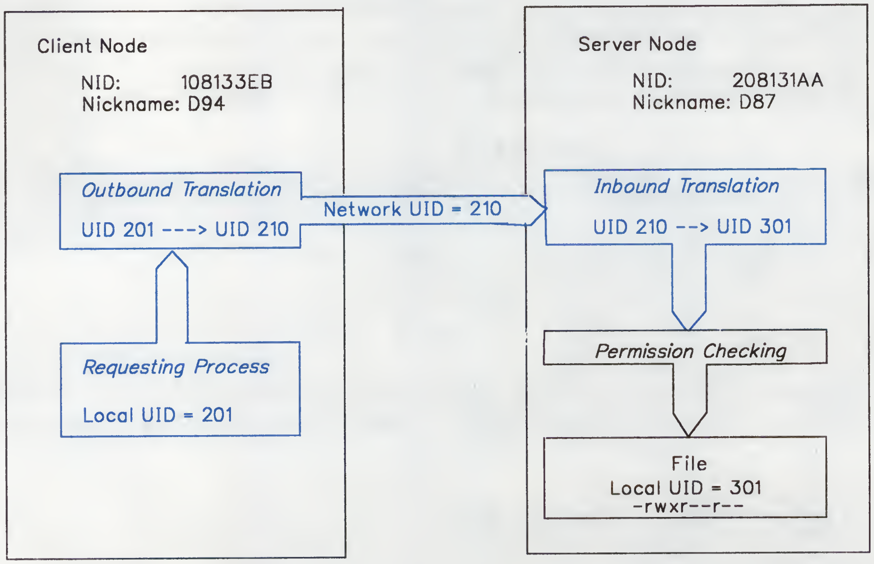


Figure 11-10. Outbound and Inbound ID Translation

When used together, outbound and inbound translation can simplify certain network management tasks. For example, the same user (tom) might have a different UID on each of five nodes in a network:

User Name	Node	UID
tom	Node_A	300
tom	Node_B	214
tom	Node_C	461
tom	Node_D	302
tom	Node_E	270

Without outbound translation, each of the five lines in the translation table at a given node must have a different value for the inbound network UID:

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
tom	U	300	-	300	Node_A
tom	U	300	-	214	Node_B
tom	U	300	-	461	Node_C
tom	U	300	-	302	Node_D
tom	U	300	-	270	Node_E

With outbound translation, however, each node can translate the local ID to a common network ID. While the table at each node still must have five rows for the user in the example, each row has the same value for the inbound network UID (that is, you have to gather less information to set up the translation tables):

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
tom	U	300	250	250	Node_A
tom	U	300	250	250	Node_B
tom	U	300	250	250	Node_C
tom	U	300	250	250	Node_D
tom	U	300	250	250	Node_E

You can further simplify the translation tables by using wildcards in conjunction with outbound translation. For example, the five lines required for the user in the previous example can be reduced to a single line if you use a wildcard for the NID:

```
Usr/Grp           Local           Network ID         Originating Node
 Name     U/G      ID       Outbound       Inbound    Name/Nickname
-----------------------------------------------------------------------

 tom       U      300          250           250      *
```

In practice, a network administrator should maintain a registry of user names and their corresponding network IDs. Then, when a user is added to a node on the network, the person who creates the account can refer to the registry for the common network IDs.

Managing ID Translation

The Distributed Services ID translation facilities make possible five different types of individual translation entries. You can associate:

1. One inbound network ID (from one node) with one local ID.
2. One inbound network ID (from all nodes) with one local ID.
3. All inbound network IDs (from one node) with one local ID.
4. All inbound network IDs (from all nodes) with one local ID.
5. One local ID with one outbound network ID.

In addition, by having several instances of entry types 1, 3, or 5, each naming a different node, you can have a total of eight possible translation relationships, as illustrated in the following table:

Local IDs	Outbound IDs	Inbound IDs	Node IDs
One	One	One	One
One	One	One	Several
One	One	One	All (*)
One	One	All	One
One	One	All (*)	Several
One	One	All (*)	All (*)

Figure 11-11 (Part 1 of 2). ID Translation: Possible Combinations

Local IDs	Outbound IDs	Inbound IDs	Node IDs
One	One		
Several	One		

Figure 11-11 (Part 2 of 2). ID Translation: Possible Combinations

Among all these possibilities, how do you choose the ones to use in your own translation strategy? The following are three general guidelines to follow when planning and managing ID translation:

1. Make read-only access the "default" access granted at each node.
   - To avoid automatically granting write access to unknown users, do not add both wildcard GID and wildcard UID translation entries to the Network Users/Groups Table.
   - Add UID translation entries only for those network users with write access to the node. Network users do not need a translated user ID for read-only access.
2. Use UID translation entries to control write access:
   - Map one network user to one local ID.

     Do not map several network IDs to one local ID unless all IDs belong to the same network user (and therefore will not be simultaneously active on the network).
   - Create a local user ID for each network user with write access to the node.

     Do not map a network user to an already existing local ID unless both IDs really belong to the same user.
3. Use wildcard NID entries with caution on a large network (especially one in which you do not have customizing authority on all nodes).

Additional Guidelines

- Use one-to-one translation entries to grant write access.

  This lessens the chances of having more than one user ID "own" a file. You should minimize the possibility of several users writing simultaneously to the same file, unless they are using applications specifically designed to manage such access. Otherwise, the changes made by one user may be lost.

  For example, if two **ed** editing sessions have the same remote file open simultaneously, only the changes made by the session that ends first are saved. If the work from both editing sessions is to be saved, the first one to end should write the file to a different file name, the second one to the original file name. This strategy assumes, of course,

that the first user knows that the second has the file open. One way to reduce the possibility of such conflicts is to use relatively restrictive file permissions. Another is to use restrictive ID translation. Using both restrictive ID translation and restrictive file permissions (for example, write only by owner) is probably the safest strategy to follow.

- Grant read-only access to users who only need to access remote commands.
- Grant read-only access to unknown users, especially if you are not responsible for adding new users to all the nodes in the network. For nodes that you have control over, you might have both a global UID and a global GID entry for your own (temporary) convenience.
- As a general rule, grant read-only access (or no access) to any remote ID that is not normally used as a login ID.
- Consider carefully the potential consequences of granting read, write, or no access to system IDs such as **root**, **system**, **mail**, and those IDs belonging to system daemons (**tftpd** and **smtpd**, for example). You may need to find out how the IDs installed by some licensed programs (such as TCP/IP) are used by these programs to judge whether or not they need to be included in your translation plan.

If you want to run remote commands that run in either set-user-ID or set-group-ID mode, the permissions of those commands and the files that it uses must be set to allow read by others. Commands with read by other permission set are transferred to the remote system before permission checking is done, so that you do not need to translate user ID 0 to user ID 0 for all systems.

If you want to perform remote network administrative tasks, you must translate an incoming Group ID to GID **0** at the nodes you want to customize. This does not, however, mean that you must translate GID **0** to GID **0**, just that some group on the server must translate to **system** group on the client. If you do translate GID **0** to GID **0**, any network user that is a member of **system** group on his/her local system can, for example, change the network tables on your system.

One conservative approach to implementing remote administration is to create a special ID that can be mapped to user **root** or a special group that can be mapped to **system**. Be sure to limit access to the **/etc/passwd** and **/etc/group** files and the **users** command on the system where this login ID or group resides.

Finally, if you do not map either **root** or **system** to some local ID (**guest** or **usr**, for example), network nodes may not be able to issue mounts automatically at system start-up, since these are normally done by **root** through the **/etc/rc.ds** initialization file.

Using the dsldxprof Command

If you to create an ASCII file that defines user and group information in a special format, the **dsldxprof** command allows you to load that file into the Network Users/Groups Table. Using this command you can:

- Load initial or additional values into a Network Users/Groups Table
- Delete entries from a Network Users/Groups Table
- Change entries in a Network Users/Groups Table.

The procedures for these tasks are described in "Working with the Network Users/Groups Table" on page 11-107. Refer to *AIX Operating System Commands Reference* for complete information about this command. This command is useful to work with large Network Users/Groups Tables or to many changes. In addition, to build several Network Users/Groups Tables that have many translation entries in common, you can define those entries in a single ASCII file, and use **dsldxprof** to load them into each Network Users/Groups Table.

Input Format

The **dsldxprof** command reads one line at a time from standard input or from an ASCII file specified with the **-f** flag. Each input line defines one translation entry and must contain the following fields:

name *type* *local-ID* *outbound-ID* *inbound-ID* *node-ID* *comments*

where

name Specifies a login name. If you provide a local ID number in the third field, you can use a - (minus) character as a place holder for this field. This field cannot begin with * (asterisk, the comment character).

type Specifies whether the line defines a UID or a GID translation entry. The character **U** specifies a UID translation entry; the character **G** a GID translation entry. These can be lower- or uppercase characters.

local-ID Specifies a local ID number. If you specify a valid user name in the first field, you can use a - (minus) character as a place holder for this field.

Note: Unlike the **ugtable** command, **dsldxprof** does not warn you if the ID number is not defined in the **/etc/passwd** or **/etc/group** file.

outbound-ID Specifies the outbound network ID number to associate with the local ID number. If you specify an inbound ID and a node ID, you can use a - (minus) character as a place holder for this field.

inbound-ID Specifies the incoming network ID number to associate with the local ID number. If you do not specify a node ID, you can use a - (minus) character as a place holder for this field. If you do specify a node ID, you must provide a value for this field.

node-ID Specifies the node ID number or nickname to associate with the inbound ID number specified in field four. If you do not specify an inbound ID, you can use a - (minus) character as a place holder for this field.

comments Any information appearing after the sixth field is ignored. Use this space for comments, as needed. For more extensive comments, begin each line that you want ignored with * (asterisk).

For example, the following input line:

```
-       U       0       0       0       *
```

specifies the same translation entry as the following **ugtable** panel:

```
User/Group Entry                   »User  »Group

User/Group Local ID                »0.....
(opt. if U/G name entered exists)

User/Group Name                    ».......
(Ignored if U/G Id entered)

Outbound Network ID                »0.........
Inbound Network ID                 »0.........

Originating Node ID/Nickname       »*............
```

As another example, the following file defines the translation entries necessary to set up limited network access for most users and to allow one network user to have local superuser authority:

```
-    U    100    100    *      *
-    G    100    100    *      *
-    U    0      -      201    20810CBF
```

If this file is named `idtable` and is copied to each node in the network, the following command:

```
# dsldxprof -f idtable
```

builds a Network Users/Groups Table that enables user 201 working at node 20810CBF to do any further network customization needed at any node in the network. Other users have limited access to the network until that customization is completed.

Note: The Distributed Network Node Tables must also be built at each node before user 201 can do remote customization.

Using dsldxprof with Remote Tables

You can add, delete, and modify translation entries in a remote Network Users/Groups Table by running **dsldxprof** with the **-n** flag, for example:

```
# dsldxprof -n 20810CBF -f rmtable
```

In order to modify a remote table, you must have **system** group or superuser privileges at the remote node. In addition, the **dsxlate** command must be run at that node before your changes take effect.

Working with the Distributed IPC Queues Table

The ***Distributed IPC Queues Table*** defines the IPC queues (both local and remote) that are known to the local node. Each entry in the table identifies the queue name, the local key, the remote key and the node ID or nickname of the system on which the queue is implemented. This information is stored in profiles.

Use the **ipctable** command to access the Distributed IPC Queues Table.

Starting the ipctable command

1. Enter the **ipctable** command on the command line:

   ```
   ipctable
   ```

   A pop-up panel appears requesting you to identify the node.

   ```
   Enter Node ID Nickname        ».............
   (Default is the local id.)
   ```

   If you specify a NID or node nickname, the files found in **/etc/profsvcs/pfslocal** on the specified node are accessed. If you specify a directory name, the command looks under **/etc/profsvcs** for a directory with that name and uses the files found in **etc/profsvcs/pfslocal** under that directory. The default value is the local node ID. Profiles for the local node are in **/etc/profsvcs/pfslocal**.

2. Press **Do** when you have entered the correct value. The Distributed IPC Queues Table appears.
3. Once the table is displayed, you can add, delete or change the information in that table.
4. Select **Close** on the command bar to end the session.

Adding a Message Queue to the Network

Selecting ADD allows you to add a new entry to the list.

When you select ADD from the command bar of the Distributed IPC Queues Table window, you will see the following pop-up panel:

```
Queue Name                 ».............
Local Key                  »......      (196608-104575)
Remote Key                 »......      (196608-104575)
Remote Node ID/Nickname    ».............
```

If you are adding a local queue profile, you must specify a Queue Name. You can also specify a Local Key (one not already assigned). If you do not specify a local key, **ipctable** provides that value when you press **Do**.

If you are adding a remote queue profile, you must specify a Queue Name, a Remote Key, and a Remote Node ID/Nickname. The queue name and remote key must match those of the corresponding queue profile installed at the remote node.

If you enter a NID in the Originating Node ID/Nickname input field, you must enter all alphabetic characters as uppercase characters. Unlike **ndtable**, **ipctable** cannot convert lowercase characters to uppercase, since you can enter a node nickname in the same field, a name that can contain both upper- and lowercase characters.

Changing IPC Message Queues

Selecting CHANGE allows you to modify the description of an existing entry.

You can change the local queue name, the local IPC key, the remote IPC key, or the remote node ID or nickname.

Deleting an IPC Message Queue

Warning: The `DELETE` command permanently erases the selected profile entry.

Selecting `DELETE` allows you to remove the IPC queue that you have selected from the table.

When you select `DELETE` from the command bar, a pop-up panel appears that displays the name of the selected queue. Press **Do** to delete the queue. When you delete an IPC queue entry in the local IPC table, the kernel rebuilds its IPC table when you exit **ipctable**.

Configuring Remote Profiles

To work with the profiles on a remote node:

- The remote node must be defined in the local Distributed Network Node Table, and the local node must be defined in the remote Distributed Network Node Table.
- You must be a member of the local **system** group or be running with local superuser authority. You must also have **system** group or superuser access to the remote node.
- When you use **ugtable**, **dsldxprof**, or **ipctable** to create or modify remote tables, the command does not load the corresponding table into the remote kernel when you exit. Someone must run **dsxlate** or **dsipc** at the remote node to load the tables into the remote kernel.

Enabling Remote Configuration

To be able to customize a remote node, follow these steps before doing any other customization:

To Allow Remote Configuration from One Node

1. Select one node to be the administrative node.
2. At the administrative node, add Distributed Network Node Table entries describing each node in the network.
3. At each nonadministrative node, add a Distributed Network Node Table entry describing the administrative node.
4. At each nonadministrative node, add an entry to the Network Users/Groups Table that translates group 0 from the administrative node to local group 0

Changing Remote Profiles

To change the profiles on a remote node, enter the NID or nickname of that node in the pop-up panel that appears when you run **ndtable**, **pwtable**, **ugtable**, or **ipctable**:

```
Enter Node ID Nickname       ».............
(Default is the local id.)
```

This causes the command to mount the remote **/etc/profsvcs/pfslocal** directory and allows you to modify the profiles in that remote directory. You can access and modify these profiles just as you do local profiles with the following differences:

- If you are not a member of the **system** group or do not have superuser authority, you cannot browse remote tables.
- **ugtable** and **ipctable** do not load the Network Users/Groups Table or Distributed IPC Queues Table into a remote kernel when you end the command.

  Note that any changes you make to the Network Users/Groups Table or Distributed IPC Queues Table do not take effect until the tables are loaded into the kernel. To load the tables, do one of the following:

  - Have a user at the remote node shut down and restart Distributed Services (see "Stopping Distributed Services" on page 11-129 and "Restarting Distributed Services" on page 11-130).
  - Run the **dsxlate** or **dsipc** command at the remote node.

    If you have installed TCP/IP and have a remote login ID with superuser or system group authority, you may be able to use the **tn** command to log onto the remote node or the **rexec** command to run **dsxlate** remotely. See *Interface Program for use with TCP/IP* for more information on running these commands.

Maintaining Distributed Services

Once it is properly configured, Distributed Services operates without intervention. Maintenance is required to help diagnose network problems and to perform routine backup and restore procedures to protect the information in the profiles. The following procedures guide you in these tasks.

Stopping Distributed Services

Before performing maintenance tasks that affect the Distributed Services profiles or executable files, stop both Distributed Services and SNA Services as outlined in the following procedure.

Stopping Distributed Services

1. Use the **shutdown -d** command to stop Distributed Services:

   ```
   shutdown -d
   ```

2. Stop SNA Services:

   ```
   stop sna
   ```

More Detailed Information

1. Shut down Distributed Services:

   ```
   # shutdown -d
   ```

   When you specify the **-d** flag, **shutdown** issues the following commands (after sending a warning message to logged-on users and waiting 60 seconds):

   ```
   /etc/dsstate -sb
   /etc/umount allr
   /etc/dsstate -ab
   ```

   The two **dsstate** commands block all incoming and outgoing network activity. The **umount** command unmounts all remote directories.

   Note: If you have mounted local files or directories over remote directories, you must unmount the local object before the **umount** command can unmount all remote directories.

   Shutting down Distributed Services stops network traffic in and out of the node, but it does not shut down SNA Services.

If you are the only user currently logged on, or if there is no current remote activity, you can use the “fast” version of **shutdown**:

```
# shutdown -df
```

When you specify both the **-d** and **-f** flags, **shutdown** does not send a warning message to other logged-on users and does not wait 60 seconds before issuing the **dsstate** commands or the **umount** command. Alternatively, you can specify some period other than 60 seconds (instead of specifying **-f**), for example:

```
# shutdown -d 15
```

This sends the warning message and waits 15 seconds before proceeding.

2. Stop SNA Services:

   ```
   # stop sna
   ```

   This step is necessary because AIX will not copy new versions of the SNA profiles over the files currently in use. If you are not restoring profiles into the **/etc/profsvcs/pfslocal** directory, you do not need to shut down Distributed Services and stop SNA Services. (See “Creating an Alternate Set of Local Profiles” on page 11-135 for more information about alternate profiles.)

Restarting Distributed Services

Use the following procedure to restart Distributed Services and SNA Services after you have completed maintenance procedures.

Restarting Distributed Services

1. Run the Distributed Services startup file **/etc/rc.ds** to restart both Distributed Services and SNA Services.

   ```
   # rc.ds
   ```

More Detailed Information

The startup file **/etc/rc.ds** is a file that the system runs when it starts up. You can also run it directly from the command line to restart Distributed Services and SNA Services:

```
# rc.ds
```

This shell program:

- Runs **dsxlate** to load the Distributed Network Node Table into the kernel
- Runs **dsipc** to load the Distributed IPC Queues Table into the kernel

- Starts SNA Services
- Starts default attachments that you have specified
- Runs **dsstate** to set the Distributed Services kernel logic. See the **dsstate** command in *AIX Operating System Commands Reference* for a discussion of all **dsstate** flags.

Backing Up Distributed Services Profiles

You should periodically make a backup copy of the Distributed Services profiles to protect against loss of these profiles.

To Back Up Distributed Services Profiles

1. Change directories to the directory in which the profiles reside:

```
# cd /etc/profsvcs/pfslocal
```

2. Enter the following **backup** command:

```
# ls -a | backup -iv
```

More Detailed Information

In addition to the normal periodic backup procedure, back up the profiles under any of the following conditions:

- Before and after you create a new set of profiles.
- Before and after you update or change a current set of profiles.
- Before and after you change the security level of SNA Services.
- Before and after you change the communications password (to protect against forgetting the password).

Enter the following commands on the AIX Operating System command line to back up the Distributed Services profiles in the standard directory:

```
# cd /etc/profsvcs/pfslocal
# ls -a ¦ backup -iv
```

To back up profiles in a different directory, use the **cd** command to change to that directory before using the **backup** command. This procedure copies all of the profiles in the selected directory to the diskette in **/dev/fd0**. If you use another method of backing up the files, make sure that the method you choose backs up the hidden files (file names beginning with a . [period]) as well as the normal files.

Restoring a Backup Copy of Distributed Services Profiles

To Restore Distributed Services Profiles

1. Stop Distributed Services (see "Stopping Distributed Services" on page 11-129).
2. Change to the directory in which the Distributed Services profiles reside:

   ```
   # cd /etc/profsvcs/pfslocal
   ```

3. Insert the backup diskette in diskette drive 1 and enter the following **restore** command:

   ```
   # restore -xv
   ```

4. Restart Distributed Services (see "Restarting Distributed Services" on page 11-130).

More Detailed Information

Stop Distributed Services (see "Stopping Distributed Services" on page 11-129) and change to the directory that will contain the restored profiles. For example, to restore Distributed Services profiles that were backed up to diskette using the procedure described in "Backing Up Distributed Services Profiles" on page 11-131:

```
# cd /etc/profsvcs/pfslocal
```

Restore the files:

```
# restore -xv
```

Restart Distributed Services as described in "Restarting Distributed Services" on page 11-130.

Recovering Distributed Services Profiles

Infrequently, you may receive an error message telling you to recover your profiles. When you receive this message, or any time that you suspect that your profiles may be damaged, run the **/etc/profsvcs/profck** command as follows:

```
# /etc/profsvcs/profck  -ds  /etc/profsvcs/pfslocal
```

This command tries to rebuild the profiles in the directory you specify. If **profck** cannot rebuild the profiles, restore a backup copy of the profiles (see "Restoring a Backup Copy of Distributed Services Profiles").

You can also use the **recover** command to rebuild individual database files as follows:

```
# /etc/profsvcs/recover  /etc/profsvcs/pfslocal/pfsnid
# /etc/profsvcs/recover  /etc/profsvcs/pfslocal/pfsuidgid
# /etc/profsvcs/recover  /etc/profsvcs/pfslocal/pfsipcky
```

In this example, the first command tries to recover the node ID and nickname profiles associated with the Distributed Network Node Table. The second tries to recover the Network Users/Groups Table. The third tries to recover the Distributed IPC Queues Table.

See the discussions of the **profck** and **recover** commands in *SNA Services Guide and Reference* for more detailed information about these commands.

Creating a New Set of Default Profiles

If your profile data base becomes unusable, use the following procedures to create a new set of default profiles.

Making a New Set of Default Profiles

The profiles that Distributed Services and SNA Services use are in the directory **/etc/profsvcs/pfslocal**. If this directory is not empty, you will get error messages from **peu** and **dspeu** and the new profiles will not be created.

1. Stop Distributed Services and SNA Services (see "Stopping Distributed Services" on page 11-129).
2. Remove all files from **/etc/profsvcs/pfslocal**:

   ```
   rm /etc/profsvcs/pfslocal/*
   rm /etc/profsvcs/pfslocal/.*
   ```

3. Run the **peu** command to create a set of default profiles for SNA Services. This command may take about a minute to complete.

   ```
   peu
   ```

4. Run the **dspeu** command to create a set of default profiles for Distributed Services.

   ```
   dspeu
   ```

The directory **/etc/profsvcs/pfslocal** now contains a new set of default profiles for Distributed Services and SNA Services. Configure the new profiles using the procedures in "Configuring a Basic Distributed Services System" on page 11-17.

To make a new set of default profiles in a different directory while keeping the original set in **/etc/profsvcs/pfslocal**, use the following procedure.

Making an Alternate Set of Default Profiles

1. Stop Distributed Services and SNA Services (see "Stopping Distributed Services" on page 11-129).
2. Change directory to **/etc/profsvcs**.
3. Make a new directory in this directory with a name that you choose.
4. Make a new directory path in this new directory to contain the new set of default profiles. The new directory name relative to the newly created
5. Run the **peu** command (using the name for the new profile directory) to create a set of default profiles for SNA Services. This command may take about a minute to complete.
6. Run the **dspeu** command (using the name for the new profile directory) to create a set of default profiles for Distributed Services.
7. Restart Distributed Services and SNA Services (see "Restarting Distributed Services" on page 11-130).

The directory **/etc/profsvcs/athena/etc/profsvcs/pfslocal** now contains a new set of default profiles for Distributed Services and SNA Services. Configure the new profiles using the procedures in "Configuring a Basic Distributed Services System" on page 11-17. When the configuration programs ask for Node ID Nickname, enter the name of the new directory (athena is this example).

More Detailed Information

1. Stop Distributed Services and SNA Services (see "Stopping Distributed Services" on page 11-129).
2. Change directory to **/etc/profsvcs**:

   ```
   # cd /etc/profsvcs
   ```

3. Make a new directory in this directory with a name that you choose. For example, if the new profiles are to be used to configure system `athena`, you might create the directory as follows:

   ```
   # mkdir athena
   ```

4. Make a new directory path in this new directory to contain the new set of default profiles. The new directory name relative to the newly created directory must be **./etc/profsvcs/pfslocal**. For example, the following instructions create that directory (while still using **/etc/profsvcs** as the current directory):

   ```
   # mkdir athena/etc
   # mkdir athena/etc/profsvcs
   # mkdir athena/etc/profsvcs/pfslocal
   ```

5. Run the **peu** command (using the name for the new profile directory) to create a set of default profiles for SNA Services. This command may take about a minute to complete. For example (while still using **/etc/profsvcs** as the current directory):

   ```
   # peu athena
   ```

6. Run the **dspeu** command (using the name for the new profile directory) to create a set of default profiles for Distributed Services. For example:

   ```
   # dspeu athena
   ```

7. Restart Distributed Services and SNA Services (see "Restarting Distributed Services" on page 11-130).

Creating an Alternate Set of Local Profiles

The default Distributed Services profiles are stored in the **/etc/profsvcs/pfslocal** directory. However, you can create additional sets of profiles for Distributed Services to use and store those sets in other directories in the directory **/etc/profsvcs**.

Creating an Alternate Set of Local Profiles

1. Create a new directory path in the **/etc/profsvcs** directory to contain the new set of profiles.
2. Copy all files (* and .*) from **/etc/profsvcs/pfslocal** into the new directory.
3. Use the Distributed Services customization programs to change the tables as required. Be sure to specify the new directory name instead of accepting the default when the system asks for the `Node ID/Nickname`.
4. Mount the new profile directory onto **/etc/profsvcs/pfslocal** when you want the system to use them.

More Detailed Information

1. Create a new directory path in the directory **/etc/profsvcs** to contain the new set of profiles (the Distributed Services customization commands expect **/etc/profsvcs** to be the parent directory.) The path name relative to **/etc/profsvcs** must be in the form:

   ./*dirname*/etc/profsvcs/pfslocal

   For example, to create a path to store profiles in a new directory called `newprofs`, use the following commands:

   ```
   # cd /etc/profsvcs
   # mkdir newprofs
   # mkdir newprofs/etc
   # mkdir newprofs/etc/profsvcs
   # mkdir newprofs/etc/profsvcs/pfslocal
   ```

2. Copy all files from **/etc/profsvcs/pfslocal** into the new directory. Note that each profile data base has an index file (with a corresponding file name beginning with a . [period]).

   If you use the AIX **cp** command, be sure to specify both the data base files and the "hidden" index files:

   ```
   # cd /etc/profsvcs/pfslocal
   # cp *  .*  /etc/profsvcs/newprofs/etc/profsvcs/pfslocal
   ```

   If you have installed Data Management Services, you can use its **copy** command. This command copies the index files along with the data base files you specify:

   ```
   # cd /etc/profsvcs/pfslocal
   # copy * /etc/profsvcs/newprofs/etc/profsvcs/pfslocal
   ```

3. Use the Distributed Services customization programs to change the profiles as required. When you run **ndtable**, **pwtable**, **ugtable**, or **ipctable**, a pop-up panel appears:

   ```
   Enter Node ID Nickname       ».............
   (Default is the local id.)
   ```

   Instead of selecting the default value, enter the name of the new directory that contains the alternate set of profiles and then press **Do**. For example, enter the directory name `newprofs` to select the new set of profiles created in the previous examples. The full path name of the directory is not required because the specified

directory must be in **/etc/profsvcs**. You can then change the values in the profiles in the new directory.

4. Mount the new profiles onto **/etc/profsvcs/pfslocal** when you want the system to use them:

   a. Stop Distributed Services and SNA Services (see "Stopping Distributed Services" on page 11-129).

   b. Mount the profiles:

   ```
   # cd /etc/profsvcs
   # mount newprofs/etc/profsvcs/pfslocal pfslocal
   mount newprofs/etc/profsvcs/pfslocal onto pfslocal ...
   Ok (warning: /etc/profsvcs/pfslocal wasn't empty)
   ```

   c. Restart Distributed Services and SNA Services (see "Restarting Distributed Services" on page 11-130).

   "Restoring a Backup Copy of Distributed Services Profiles" on page 11-132 for stopping Distributed Services and SNA Services.)

When you want to return to the previous set of profiles, unmount the new set of profiles:

1. Stop Distributed Services and SNA Services (see "Stopping Distributed Services" on page 11-129).

2. Unmount the profiles:

   ```
   # umount pfslocal
   unmounting /etc/profsvcs/pfslocal ... ok
   ```

3. Restart Distributed Services and SNA Services (see "Restarting Distributed Services" on page 11-130).

Creating an Alternate Set of Remote Profiles

Use the following procedure to create alternate sets of profiles to be used on another node:

Creating an Alternate Set of Remote Profiles

1. Create a new directory path in the **/etc/profsvcs** directory to contain the new set of profiles.
2. Mount the remote **/etc/profsvcs/pfslocal** directory onto **/mnt**.
3. Copy all files (* and .*) from **/mnt** into the new directory:
4. Unmount the remote **/etc/profsvcs/pfslocal** directory.
5. Use the Distributed Services customization programs to change the tables as required. Be sure to specify the new directory name instead of accepting the default when the system asks for the `Node ID/Nickname`.
6. Copy the new profiles to the remote **/etc/profsvcs/pfslocal** directory when you want the remote system to use them.

More Detailed Information

1. Create a new directory path in the directory **/etc/profsvcs** to contain the new set of profiles (the Distributed Services customization commands expect **/etc/profsvcs** to be the parent directory.) The path name relative to **/etc/profsvcs** must be in the form:

   ./*dirname*/etc/profsvcs/pfslocal

   For example, to create a path to store profiles in a new directory called `remprofs`, use the following commands:

   ```
   # cd /etc/profsvcs
   # mkdir remprofs
   # mkdir remprofs/etc
   # mkdir remprofs/etc/profsvcs
   # mkdir remprofs/etc/profsvcs/pfslocal
   ```

2. Mount the remote **/etc/profsvcs/pfslocal** directory in a temporary location:

   ```
   # mount -n node /etc/profsvcs/pfslocal /mnt
   ```

3. Copy all files (* and .*) from the remote directory (**/mnt**) into the new directory:

   ```
   # cd /mnt
   # cp .* * /etc/profsvcs/pfslocal/remoteprofs
   ```

4. Unmount the remote **/etc/profsvcs/pfslocal** directory:

```
# cd
# umount  /mnt
```

5. Use the Distributed Services customization programs to change the tables as required. Be sure to specify the new directory name instead of accepting the default when the system asks for the `Node ID/Nickname`.

6. Copy the new profiles to the remote **/etc/profsvcs/pfslocal** directory when you want the remote system to use them. (Follow the same procedures listed in "Stopping Distributed Services" on page 11-129 for stopping Distributed Services and SNA Services and in "Changing Remote Profiles" on page 11-128 for reloading remote profiles.)

Distributed Services Tuning and Problem Determination

In a distributed AIX environment, performance tuning and error handling are closely related. For example, in some situations, increasing the number of buffers in the device buffer pool may improve the server's performance in handling concurrent remote requests; an insufficient number of buffers can cause a connection to fail. This section describes several modifications that may improve performance, correct error conditions (such as connection failures, kernel panics, and hangs), or both.

Notes:

1. The default values for the parameters discussed here are adequate for typical network environments. Whether you will be able to improve performance by altering these values depends upon the unique characteristics and requirements of your Distributed Services installation.

2. This section does not discuss the SNA Services profile parameters that could have an effect on Distributed Services performance. For information about these profile parameters, see *SNA Services Guide and Reference.*

If you are trying to correct an error condition, you should check two things before you consider changing the values described in this section:

1. Make sure that the network hardware options are installed correctly.

2. Make sure that the remote node or nodes are running normally.

If Interface Program for use with TCP/IP is installed on your network, you can use the **ping** command to determine whether remote nodes are responding and, thus, whether the network hardware is working properly.

Increasing Kernel Buffers

The **kbuffers** parameter in the **/etc/master** file determines how many kernel buffers are reserved for disk I/O and the i-nodes of open files. Increasing the value of the **kbuffers** parameter can improve the speed of I/O operations, depending upon the amount of memory and disk space available. Beyond a certain point, however, increasing the value of **kbuffers** degrades performance.

The default value of **kbuffers (0)** reserves 100 buffers (2048 bytes each) for every M byte of memory installed. To change the value, first change the value of **kbuffers** in **/etc/master**, then generate a new kernel by following the procedure described in "Generating a New Kernel" on page 3-33.

Increasing Kernel Processes

The maximum number of concurrent, remote requests that a server can handle is limited by the maximum number of **kprocs** (internal kernel processes) that the server can run. Two conditions that may indicate an inadequate number of **kprocs** are:

- Slower performance in handling local requests
- Connection failures.

The **kprocs** parameter in **/etc/master** sets the maximum number of internal kernel processes. The default value of **kprocs** is 21. Generally, there should be two **kprocs** for each concurrent client request. Before you change the **kprocs** value, estimate the maximum number of remote requests that will run concurrently, multiply that estimate by two, and compare the result to the default (or current) value of **kprocs**.

To change the number of **kprocs** that can be run, do the following:

1. Change the value of **kprocs** in **/etc/master**.
2. Generate a new kernel (following the procedure described in "Generating a New Kernel" on page 3-33).
3. Run a **dsstate** command of the form:

`dsstate` *value*

where *value* is the value of **kprocs** in **/etc/master**; this determines the number of possible **kprocs** that can run. (See *AIX Operating System Commands Reference* for information about **dsstate**.)

Increasing Device Buffers

In a distributed AIX environment, increasing the number of device buffers may:

- Improve performance
- Increase the number of concurrent remote requests that a server can handle
- Eliminate connection failures.

To change the number of device buffers, change the values of one or more of the following network adapter parameters with the **devices** command:

nobodr Determines the number of buffers in the device ring. The default value of **nobodr** is 30.

When the number of requests to a server exceeds the capacity of the server's internal message queues, some message transmissions fail. The server continues to try to transmit those messages; consequently, its performance slows. If the server is unable to transmit the message successfully within a certain number of attempts, the transmission fails. Increasing the value of **nobodr** may restore or improve performance by reducing message retransmission.

norbosr Determines the number of receive buffers in the SLIH ring. The default value of **norbosr** is 30. The value of **norbosr** should be the same as that of the **nobodr** parameter.

nobibp Determines the number of buffers in the network device buffer pool. The default value of **nobibp** is 100. A larger value can enable a server to handle more concurrent remote requests and may eliminate some connection failures.

Use the following formula as a guide for changing the value of **nobibp**:

```
nobodr + n * (tw * 2) + 2 * n
```

where:

`nobodr` is the value of that device parameter.

`n` is the number of nodes in the network.

`tw` is the value of the SNA Services transmit window parameter. (For the value of `tw`, see the applicable SNA Services logical link profile.)

For example, if the server is on a four-node network, its **nobodr** value is 30, and its **tw** value is 16, the value for **nobibp** is 166:

```
30 + 4 * (16 * 2) + 2 * 4 = 166
```

Changing Device Descriptions

The values for the device description parameters should be set according to the configuration of your Distributed Services network. Connections fail if the values of these parameters are too small. To change one or both of the following network adapter parameters, use the **devices** command :

mnonid Sets the maximum number of node IDs that the server can handle. The default value of **mnonid** is 4. The value of **mnonid** should be equal to the number of clients on the network.

mnoal Sets the maximum number of gateways that the server can handle, including the base network. The default value of **mnoal** is 3. The value of **mnoal** should be equal to the number of network adapters installed in the server.

Tuning for Larger Networks and More Concurrent Activity

By modifying the values of the system parameters addressed in this section, you may be able to improve performance or correct error conditions on larger networks or on nodes that have unusually high concurrent file and process activity.

Note: Because each Distributed Services network is unique, there cannot be clear guidelines about what constitutes either a large network or unusually high concurrency.

The parameters covered in this section are in the **/etc/master** file. To change them, do the following:

1. Make the necessary changes in **/etc/master**.
2. Generate a new kernel (as described in "Generating a New Kernel" on page 3-33).
3. Increase the amount of paging space on the system (as described under "Increasing Paging Space" on page 11-143) to accomodate the new, larger kernel.

Parameters

dsnkprocs Sets the maximum number of kernel processes available for use by Distributed Services. The value of **dsnkprocs** should equal the number of concurrent connections. The default value of **dsnkprocs** is 20.

nncb Sets the amount of memory allocated for Distributed Services translation tables. The value of **nncb** should equal the number of concurrent connections. If **nncb** is inadequate, the **dsxlate** command will fail. The default value of **nncb** is 25.

maxnode Sets the maximum number of nodes that can be connected in the Distributed Services network. The default value of **maxnode** is 20.

filetab Sets the maximum number of files (local and remote) that can be open simultaneously. The default value of **filetab** is 250. The values of **filetab** and **inodetab** should be approximately equal.

inodetab Sets the maximum number of i-nodes that can be active simultaneously. The value of **inodetab** should equal the number of files (local and remote) that are open simultaneously. The default value of **inodetab** is 250. The values of **inodetab** and **filetab** should be approximately equal.

callouts Sets the maximum number of timed events that can be scheduled concurrently. The value of **callouts** should be approximately:

maxnode - dsnkprocs + 50

Increasing Paging Space

When you increase the values of the parameters covered in the previous section, you also increase the size of the kernel (**/unix**). The larger kernel requires more paging space. Therefore, besides increasing the values of the parameters and generating a new kernel, you also must increase the size of the page space minidisk.

As a general guideline for how much to increase paging space, use the size difference between the new and old kernels. Use the **size** command to determine the size of each kernel. In the following example, the size difference between the new and old kernels is 69756 bytes:

```
size /unix
482272 + 67672 + 417752 = 967696
size /unix.old
443316 + 62136 + 392488 = 897940
```

To change the size of the page space minidisk, use the procedure described in *Installing and Customizing the AIX Operating System*. Since the VRM customizing program requires sizes specified in blocks, you must divide the size difference between the two kernels (bytes) by 512 (bytes per block). In the previous example, the size difference is 69756/512, or 136 blocks. To accommodate the new page space minidisk, you may need to put it on a different fixed disk (which you can do with the VRM customizing program) or rearrange the other minidisks (as is explained in "Handling the minidisk full Condition" on page 4-31).

Using trace to Monitor Distributed Services Activities

You can use the AIX Operating System trace facility to help determine whether the system is operating properly, or to indicate the nature of a problem that you may be experiencing on the network. The trace facility consists of the following components:

trace The **trace** command starts a background process that logs system events in a log file.

/etc/trcfmt This system text file defines the format of the events logged for each program that uses the trace facility.

/etc/trcprofile A system file that you edit to select the system events to be traced.

trcstop The **trcstop** command stops the background logging process.

trcrpt The **trcrpt** command formats and displays the contents of the log file.

Each of the commands is described in *AIX Operating System Commands Reference*. The file **/etc/trcfmt** is useful only to programmers designing error reports. "Selecting Trace Events for Distributed Services" on page 11-144 describes the contents of **/etc/trcprofile** and how to use it to select Distributed Services events. Refer to *AIX Operating System*

Programming Tools and Interfaces for a description of the structure of the trace facility and how to use it with an application program.

Selecting Trace Events for Distributed Services

To select the events that you want to trace, you must edit the file **/etc/trcprofile**. This file contains a list of all event classes that can be traced. Each event class is on a separate line in the file. If the line begins with an asterisk (*), events in that class are not logged; if the line does not begin with an asterisk, events in that class are logged. For Distributed Services, the following event classes appear in the file:

```
167     remote IPC            [DS_IPC]
166     rpc DS miscellaneous  [DS_MISC]
165     rpc DS locking        [DS_LK]
164     rpc DS open/close     [DS_OC]
163     rpc DS read/write     [DS_RW]
162     rpc server sending    [DS_SS]
161     rpc client receiving  [DS_CR]
160     rpc client sending    [DS_CS]
```

Each of these values selects a set of internal trace points. These values are defined in the following table. See "Remote Operations" on page 11-150 for information about these internal operations.

Number	Hook ID	Description
167	DS_IPC	Trace calls to remote IPC queues (msgget, msgsnd, msgrcv).
166	DS_MISC	Trace miscellaneous remote operations (null, mount, mntctl, msgctl, link, remove, getattr, access, setattr, statfs, mkdir, rmdir, rename, mknod, loadtbl, fsync, sgetattr).
165	DS_LK	Trace remote file locking operations (flock).
164	DS_OC	Trace remote file open and close operations (lookup, open, close, create, chg_sync, advise).
163	DS_RW	Trace remote file read and write operations (read, write, reada_clt, sync_ct, fclear, ftrunc).

Number	Hook ID	Description
162	DS_CS	The client node has sent a remote request to the server node. This event is logged in the trace report on the client node.
161	DS_SS	The server node has sent a response to a request from the client node. This event is logged in the trace report on the server node.
160	DS_CR	The client node has received a response from the server node. This event is logged in the trace report on the client node.

Tracing a Remote Mount Example

The following example shows how to use the trace facility to determine if the network is operating properly. It uses two systems that are connected by an Ethernet local area network and are running Distributed Services. These systems are:

hera Machine 2061064A (hera) is the client in this example. The client starts the remote mount operation with the command:

```
mount -n zeus /etc /mnt
```

zeus Machine 208131AA (zeus) is the server in this example. The server receives the request for the mount.

1. Before starting the **trace** program, edit the file **/etc/trcprofile** on each system to remove the asterisks from column 1 of the following lines (near the bottom of the file):

```
166     rpc DS miscellaneous  [DS_MISC]
162     rpc server sending    [DS_SS]
161     rpc client receiving  [DS_CR]
160     rpc client sending    [DS_CS]
```

   Removing the asterisks from column 1 enables tracing for the four event classes (for this example, all other lines in the file begin with an asterisk).

2. Having enabled tracing for the selected functions, start the trace program on each system with the following command:

```
$ trace
```

3. Run the **mount** command on hera to mount the **/etc** directory from zeus on the local **/mnt** directory.

```
$ mount -n zeus /etc /mnt
```

4. When the mount is complete, turn off **trace**. Enter the following command at each system:

   ```
   $ trcstop
   ```

5. Create a trace report at each node. At hera:

   ```
   $ trcrpt >hera.rpt
   ```

   At zeus:

   ```
   $ trcrpt >zeus.rpt
   ```

6. Use the **pg** command to examine the report files. The actual report may contain many other entries depending upon system activity at the time the trace was run. However, for only the remote mount, the trace report at hera might look like that shown in Figure 11-12; Figure 11-13 on page 11-147 shows the report at zeus. Figure 11-14 on page 11-148 explains the fields in the trace report.

```
                          TRACE LOG REPORT

File: /usr/adm/ras/trcfile

Wed May 13 09:42:41 1987
System: D19  1.1.2          Node: hera
Machine: 2081064A


TIME         SEQ   PID   IODN  IOCN TYPE     HOOK     DATA
09:43:42.50 0015 00215   FFFF  FFFF DS_CS    Client_s Nid=208131AA Seq#=371
       Uid=0 Gid=0 DS_Op=1
09:43:42.52 0016 00215   FFFF  FFFF DS_CR    Client_r Seq#=371 Errno=0


Summary of event counts.
rpc client sending [DS_CS]: 1
rpc client receiving [DS_CR]: 1


Total number of events: 2
```

Figure 11-12. Example Trace Report at hera

```
                         TRACE LOG REPORT

File: /usr/adm/ras/trcfile

Wed May 13 09:42:14 1987
System: D19  1.1.2          Node: zeus
Machine: 208131AA


TIME        SEQ   PID  IODN  IOCN TYPE     HOOK      DATA
09:43:00.58 0007 30002  FFFF  FFFF DS_SS    Server_s Seq#=371
        Uid=0 Gid=0 Errno=0


Summary of event counts.
rpc server sending [DS_SS]: 1


Total number of events: 1
```

Figure 11-13. Example Trace Report at zeus

Field	Definition
TIME	This field shows the time that the trace event occurred. Time is the local machine time where the event occurred. If the two machines are not exactly synchronized (as in the example), the times do not show the correct sequence of events between the two machines.
SEQ	This field contains a sequence number given to the entry in the trace log. This number is local and does not correlate across the two systems.
PID	This field contains the process ID of the process associated with the trace event. This number is the process ID on the machine that is running that process and does not correlate across the two systems.
IODN	This field is not used.
IOCN	This field is not used.
TYPE	This field defines the type of trace event that caused the entry. The entry in this field corresponds to the line in **/etc/trcprofile** from which you removed the asterisk to enable tracing that event.
HOOK	This field contains a label that identifies the trace point in the code that generated the trace entry.

Figure 11-14 (Part 1 of 2). Trace Report Field Definitions

Field	Definition
DATA	This field contains miscellaneous data that changes for each type of entry in the trace report. Each program that supports tracing has a unique set of data that it writes. The format of these reports is defined in **/etc/trcfmt**. For this example, the data included has the following meanings:
	Server_s — A server send operation caused the trace entry.
	Client_s — A client send operation caused the trace entry.
	Client_r — A client receive operation caused the trace entry.
	Seq#= — This field contains the sequence number for the operation that generated the entry in the trace report. In the example, the operation was the remote mount and the sequence number is 371. The sequence number correlates all trace entries on both systems for a given operation. That is, the number 371 is common to all trace entries (on both hera and zeus) associated with the remote mount operation.
	UID — This field contains the user ID of the process that created the trace entry. This ID is local to the system on which the trace report is generated.
	GID — This field contains the group ID of the process that created the trace entry. This ID is local to the system on which the trace report is generated.
	Errno= — This field contains the error number associated with the attempted operation. If the error number is 0, the operation was successful. Otherwise, refer to *AIX Operating System Technical Reference* **(errno.h)** for information about the indicated error.
	Nid= — This field contains the node ID of the remote system with which the operation was attempted.
	DS_Op= — This number indicates the remote operation that is associated with this entry in the trace report. The number is a displacement into the list of remote operations defined in the format definition for this entry. The format definition is found in **/etc/trcfmt**. Refer to "Remote Operations" on page 11-150 for a list of these remote operations. In Figure 11-12 on page 11-146, the value of 1 in this field indicates a dfs_mount operation.

Figure 11-14 (Part 2 of 2). Trace Report Field Definitions

Remote Operations

When Distributed Services performs operations on one system that were requested on another system, the local system (client) sends a request on the network for an operation to be performed on the remote system (server). These requests are called ***remote procedure calls***, or ***rpc***s. These operations are performed internal to the kernel and cannot be affected by the administrator or application programmer. To help understand the information contained in the trace report, the following table lists the remote operations, their DS_Op value (displacement into the list of rpc's in **/etc/trcfmt**) and a summary of the operation being requested.

DS_Op	RPC Name	Function Requested
0	dfs_null	Used to test the connection.
1	dfs_mount	Mount a remote file or directory.
2	dfs_lookup	Get status information about a remote file.
3	dfs_open	Open a remote file.
4	dfs_close	Close a remote file.
5	dfs_create	Create a remote file.
6	dfs_read	Read from a remote file.
7	dfs_write	Write to a remote file.
8	dfs_chsync	Change the sync mode of a remote file.
9	dfs_mntctl	Execute the **mntctl** system call on the remote system.
10	dfs_msgget	Execute the **msgget** system call on the remote system.
11	dfs_msgsnd	Execute the **msgsnd** system call on the remote system.
12	dfs_msgrcv	Execute the **msgrcv** system call on the remote system.
13	dfs_msgctl	Control and status for a remote message queue.
14	dfs_link	Create a **link** to a remote file.
15	dfs_remove	Remove a link to a remote file.
16	dfs_getattr	Get the attributes of a remote file.
17	dfs_access	Execute the **access** system call on the remote system.
18	dfs_setattr	Set or change attributes for a remote file or directory.
19	dfs_statfs	Execute the **ustat** system call on the remote system.
20	dfs_mkdir	Create a directory in the remote file system.
21	dfs_rmdir	Remove a remote directory.
22	dfs_rename	Rename a remote file or directory.
23	reada_clt	Remote file read-ahead.

DS_Op	RPC Name	Function Requested
24	rbsync_clt	Remote file write-behind.
25	dfs_advise	Control of the remote directory cache.
26	dfs_mknod	Execute the **mknod** system call on the remote system.
27	dfs_flock	Lock all or part of a remote file.
28	dfs_loadtbl	Execute the **loadtbl** system call on the remote system.
29	dfs_fclear	Execute the **fclear** system call on the remote system.
30	dfs_fsync	Write all data in the remote (server) buffer to disk.
31	dfs_ftruncate	Execute the **ftruncate** system call on a remote file.

Chapter 12. Managing the IBM AIX/RT Network File System

CONTENTS

About This Chapter

The IBM AIX/RT Network File System[1] is a distributed file service that allows users to share files with computers in networks that include a variety of machines and operating systems. This chapter provides the information you need to install and manage the Network File System software. It includes the following information:

- Software and hardware required to install and use the Network File System program
- What the Network File System program is and how it works
- The Network File System program software components (NFS and Yellow Pages) and what they do
- How to install the Network File System program
- How to configure the NFS software component
- How to configure and use the Yellow Pages (YP) network service.

---

1 The Network File System was developed by Sun Microsystems, Inc. NFS is a trademark of Sun Microsystems, Inc.

Information Requirements

Before reading about the Network File System, it is important that you understand the principles of AIX system management described in the following topics:

- “The File System—Background for System Management” on page 1-6 and “The Base AIX File System” on page 1-12
- “Managing User Accounts” on page 2-13
- “Information about File Systems—The /etc/filesystems File” on page 2-36
- “Backing up Files and File Systems” on page 2-45

It is also important that you understand the principles of remote communications using Transmission Control Protocol/Internet Protocol (TCP/IP). See the section on general information in *Interface Program for use with TCP/IP*.

For information on adding RT work stations to a communications network, see the section on communications planning in the *Planning Guide*.

Overview of Hardware and Software Required for the IBM AIX/RT Network File System

Before you can install the Network File System program, the following software and hardware must be installed and running on each IBM RT system that is to be on the network:

Software

- IBM RT AIX Operating System licensed program, Version 2.2.1
- IBM RT Interface Program for use with TCP/IP (Transmission Control Protocol/Internet Protocol), Version 2.2.1
- IBM RT VRM Baseband Adapter Device Driver, Version 2.2.1.

— on network for NFS product only

Hardware

- IBM RT Baseband Adapter for use with Ethernet[2]
- Cables and connectors.

Refer to the following publications for information about installing the software and hardware:

- *IBM RT Installing and Customizing the AIX Operating System*
- *IBM RT Options Installation*
- *IBM RT Planning Guide*
- *IBM RT User Setup Guide.*

---

2 Ethernet is a trademark of Xerox, Inc.

Overview of the IBM AIX/RT Network File System

The IBM AIX/RT Network File System is a distributed file system that allows users to access files located on remote computers in the network. With the Network File System program, users can share files and have access to additional disk space.

The computers that make their files and resources available for remote access are called ***servers***. The computers (or the processes they run) that access the remote resources are considered ***clients***. A computer can be both a server and a client in the Network File System environment.

The IBM AIX/RT Network File System program contains two separately installable software components, NFS and Yellow Pages (YP).

The NFS Component

When the NFS component is installed, files located on network servers can be accessed through the **mount** command. Unlike remote copy or transfer functions, users are presented with complete file systems that contain the actual files instead of copies of the files. They can read and write to the remote files as if they were located on their local machine.

How NFS Works

NFS is designed to work across networks without being dependent on machine types, operating systems, or network architectures. It achieves this independence through the use of ***remote procedure calls*** defined through the ***Remote Procedure Call*** protocol built on top of an ***eXternal Data Representation*** protocol.

NFS uses remote procedure calls to communicate between the computer on which the user is logged (the client) and the computer on which the file is located (the server). Using remote procedure calls enables users to work with remote files as if they were local. The remote procedure calls are defined through routines contained in the Remote Procedure Call (RPC) protocol. RPC makes it possible for the client to invoke a caller process that can have a process located on the server execute the procedure call as if the caller process had executed the call in its own address space. See *AIX Operating System Technical Reference* and *AIX Operating System Programming Tools and Interfaces* for more detailed information on RPC and the RPC routines.

The eXternal Data Representation (XDR) protocol provides a uniform way of representing data types over a network. The XDR routines are used by RPC to standardize the representation of data in remote communications. This resolves situations, such as different byte ordering, that can occur between communicating machines. XDR converts the parameters and results of each RPC service provided. See *AIX Operating System Technical Reference* for more detailed information on XDR and the XDR routines.

NFS uses authentication mechanisms built into the RPC facility for AIX file protection. ***Authentication*** is a means of verifying the user of an information system or resource. RPC includes a slot for the authentication parameters on every remote procedure call so that the caller can identify itself to the server. See *AIX Operating System Technical Reference* for more detailed information on the authentication routines.

NFS and RPC are not dependent on the data transport used to carry remote procedure call data. In the IBM AIX/RT Network File System environment, NFS and RPC use User Datagram Protocol/Internet Protocol (UDP/IP) and Transmission Control Protocol/Internet Protocol (TCP/IP) as the transport protocols for carrying the message data between communicating programs. UDP/IP is the default protocol used by RPC, but the default value can be changed to TCP/IP in RPC routines when necessary. See *AIX Operating System Programming Tools and Interfaces* for information on changing the default values for RPC routines.

Using NFS for Remote File Service

The NFS interface is designed to provide ***stateless*** file service. In a ***stateless*** file service environment, the server does not have to keep track of its transactions with clients, completed transactions, or files operated on from one transaction to the next. Because servers do not have to keep track of the state of each file, there is no **open** operation in NFS itself. NFS relies on the client to remember the information for later NFS uses.

An example of a stateless server operation is in the use of the **mount** command. A client can use the **mount** command to build a view of a file system located on its local devices. The client can use the **mount** command with NFS extended capabilities to build a view of a file system located on the network. This is a stateless operation because the command gets only the information needed for the client to establish contact with a server. It does not require the server to keep any information.

An NFS server can also be a client of another server. Servers can only answer requests about their own remote mounts. They cannot serve as messengers between a client and another server.

Users can use AIX commands to read and write to remote files, and to read and set file attributes on remote files. Traditional AIX commands can also be used to read remote directories, and to create and remove remote files.

IBM AIX/RT Network File System does not support access to remote special files (including the device drivers in **/dev**) or the lock manager for locking remote files.

The Yellow Pages Component

Included in the IBM AIX/RT Network File System software is the Yellow Pages, commonly referred to as the YP. The YP is a service used to administer system information on networked machines. The YP is structured as a centralized network look-up facility comprising YP data bases that provide system information, such as passwords and host names, to Network File System users.

YP Domains and Hosts

A group of hosts that share a YP data base belong to the same domain. The hosts are usually grouped together in the domain for a common reason, such as belonging to members of the same work group at a particular location. Each YP host is assigned to a domain when the system starts. The default value for the domain is specified in the **/etc/rc.nfs** file. The person who manages your system (or a user with superuser authority) can assign a host to a different domain or change the name of the domain with the **domainname** command. The domain name must be set on all host that intend to use the YP. See *AIX Operating System Commands Reference* for information on the **domainname** command.

The host that maintains the YP data base for a domain is called the ***YP master server***. To balance the YP processing task load as well as provide services in the event the YP master server is unavailable, additional hosts can be designated as ***YP slave servers.*** The YP slave servers maintain exact replicas of the master YP data base. Changes to the YP data base are made on the YP master server, and then ***propagated*** (transferred) to the YP slave servers.

Note: To ensure integrity and reliability of YP data throughout the domain, the YP service algorithm requires that changes to the data base be made only on the YP master server.

An application's access to data served by the YP is independent of the relative locations of a YP client and server. A YP client accesses system information maintained in the YP data base by making a remote procedure call to a YP server. The YP server searches its local data base and returns the requested information to the YP client.

YP Maps

Data base information is maintained in ***YP maps***. Each YP map is created by associating an index ***key*** with a ***value***. For example, the information in the YP master server's **/etc/hosts** file is used to create a YP map that uses each host name as a key. The value associated with each key contains the name, aliases, and Internet address of the host. The key and value pairs, also known as ***records***, that are created from the entries in **/etc/hosts** comprise the YP host name map.

The YP maps are created using the **makedbm** utility, which converts input into dbm format files. A YP map consists of two dbm files: *map.key*.**pag** and *map.key*.**dir**. For example, the host name map in IBM AIX/RT Network File System consists of files called

hosts.nam.pag and **hosts.nam.dir**. The file with the **.pag** extenstion contains the key and value pairs, while the file with the **.dir** extension serves as an index for large **.pag** files. The files are stored in a directory under the **/etc/yp** directory that corresponds to the name of the domain they serve. For example, maps for a domain known as `publications` would be found in the `/etc/yp/publications` directory.

Note: The **/etc/yp/YP_MAP_XLATE** table is used to resolve map naming convention differences among systems. For example, the host name map in IBM AIX/RT Network File System **hosts.nam**, is converted to **hosts.byname** for use on other systems. This table can be appended to include entries for maps specific to your site. See *AIX Operating System Technical Reference* for information on the **/etc/yp/YP_MAP_XLATE** table.

Default YP Maps

Program routines are automatically redirected from a local ASCII file to the domain-based YP map for most information maintained by the default YP maps. Routines that check password or group information can be directed to consult the local ASCII file before consulting the YP map.

The default YP maps maintain values in the same format as the ASCII files from which they are derived. For example, each entry in the **/etc/passwd** file has seven fields that are separated by colons. The values in the YP password name map have the same seven fields in the same order and format (colons separating the fields). When programs are redirected to the password name map, they find the data in the same format as the **/etc/passwd** file.

Components of IBM AIX/RT Network File System

The system calls, daemon programs, utility functions, and commands installed with each component of the IBM AIX/RT Network File System (NFS and Yellow Pages) are discussed in the following section.

NFS System Calls, Utilities, and Commands

The **async_daemon** and **nfssvc** system calls are added to the kernel when NFS is installed. The **async_daemon** system call performs asynchronous block input/output operations for NFS clients. The **nfssvc** system call loops continuously in the server to perform the server operations. See *AIX Operating System Technical Reference* for more information on these system calls.

Installing NFS causes the set of routines in the **librpcsvc.a** file to be added to the **/usr/lib** directory. The routines provide a procedure-oriented interface for calling the remote services and returning information. The routines are available to programmers. See *AIX Operating System Technical Reference* for more information on these routines.

The **rpc** file that contains the symbolic names for RPC program numbers is added to the **/etc** directory. This file provides users with readable information about the services available at RPC servers. See *AIX Operating System Technical Reference* for more information on this file.

The NFS daemons and utility programs handle the remote file system mounts and remote file operations, as well as other general network services. Some of the daemons are added to the **/etc** directory, while others are added to the **/usr/etc** directory. The daemons are started from the command line or by a system process, such as the **inetd** daemon. They can also be invoked from a start-up shell script, such as **/etc/rc.nfs**. Once started, they run continuously on the server awaiting service requests. A brief description of each NFS and RPC daemon follows:

Started from the rc.nfs file:

nfsd Network file system daemons that handle client file system requests.

biod Block input/output daemons that asynchronously process data contained in buffer caches to facilitate the **read-ahead** and **write-behind** processes in response to NFS client requests. Asynchronous blocking of data into caches increases the access speed of requests for the information contained in the buffers.

yppasswdd Updates passwords in the Yellow Pages password data base. See "Starting the yppasswdd Daemon" on page 12-44 for detailed information.

Started from the rc.tcpip file:

portmap Maps RPC program numbers to port numbers. When a client makes a remote procedure call to a specific program, it contacts the **portmap** process (also known as the **portmapper**) on the NFS server to determine the port where the request should be sent.

Started from the inetd daemon (in the inetd.conf file):

mountd Answers client file system mount requests.

pcnfsd Provides authentication and print spooling services for PC-NFS clients. This daemon can also be started from the command line. See "Configuring pcnfsd" on page 12-28 for information on configuring this daemon.

rexd Answers remote requests for execution of programs.

rstatd Returns performance statistics obtained from the kernel.

rusersd Services requests for information on system users.

sprayd Records status of data packets sent over the network by the **spray** command.

rwalld Services requests to write messages to network users.

See *AIX Operating System Commands Reference* for information specific to each daemon.

The NFS commands can be used to obtain information about remote files and to control remote file system operations made in the network. A brief description of each command follows:

nfsstat Lists statistics about NFS and RPC. Users with superuser authority can use this command to reinitialize the statistics registers.

on Runs a command on a remote computer. The superuser cannot use the **on** command.

rpcgen Compiles the input to remote procedure calls into the C language code for implementation.

rup Lists the host status of remote machines.

rusers Lists users logged in on remote machines.

rwall Writes messages to all users over the network.

showmount Lists clients that have remotely mounted a file system from a specific host.

spray Sends a stream of data packets to a specified host machine to test the system.

See *AIX Operating System Commands Reference* for more information on each command.

Yellow Pages Software Component

When the YP is running, client requests for system information are actually broadcast on the network and answered by the YP. The YP service looks up the information in the data base located on a YP server, and returns the information to the client.

The YP user and administrative commands are added to the appropriate directories when the YP is installed. A brief description of each command follows:

domainname Lists the name of the current YP domain system for a YP host. A user with superuser authority can use this command to change and set the domain.

makedbm Creates the YP data base maps.

ypbind Establishes a connection that enables a YP client process to communicate with the **ypserv** process.

ypcat Lists the contents of YP maps.

ypinit Builds and installs the YP data base on the YP servers.

yppasswd Allows users to change YP passwords from any YP host.

yppoll Identifies the version of a YP map on a YP server.

yppush Forces propagation of a changed YP map from the YP master server to each YP slave server.

ypserv Looks up information in the YP data bases.

ypset Points the **ypbind** process at a particular server.

ypwhich Identifies which machine is the YP server of a YP client.

ypxfr Requests propagation of a YP map from the YP master server. (This command is issued from YP slave servers.)

See *AIX Operating System Commands Reference* for more detailed information on each YP command.

The YP utilities that handle the functions involved with the client requests are located in **libc.a**. A brief description of each YP utility and its function follows:

Enumerating Map Values

yp_all Returns all values in the map in a single request by transferring the entire map from server to a client.

yp_first Returns the first key-value pair from a map.

yp_next Returns the next key-value pair in a map. Attributes can be set to determine how many times **yp_next** is read.

Working With Domains and Maps

yp_get_default_domain	Returns the name of a node's domain.
yp_master	Returns the machine name of the YP master server for a map.
yp_match	Returns the value associated with a passed key.
yp_order	Returns the order number for a map. An order number is the date (in UNIX seconds) the map was built.

Error Information

yperr_string	Returns a pointer to an error message string that is null-terminated but contains no period or newline character.
ypprot_err	Translates a YP protocol error code into a readable error code. The error code can be used as input to the **yperr_string** routine.

See *AIX Operating System Technical Reference* for more detailed information on the YP client utilities.

Installing the IBM AIX/RT Network File System

IBM AIX/RT Network File System is shipped as a licensed program. The installation procedures conform to the standards and conventions for all AIX licensed programs as described in the *AIX Operating System Programming Tools and Interfaces*. Be sure no other system activity is going on when the program is being installed on your system.

The NFS component can be installed by itself, or with the Yellow Pages (YP) component. It is also possible to install and use the Yellow Pages by itself.

Starting a Network File System installation

1. To start the installation, type the **installp** command at the command line.
2. Select the components to install by entering one of the following options by number:

```
1. Network File System NFS component
2. Network File System YP component
3. All of the above
```

Note: To cancel the installation, enter `quit`.

Installation of the NFS component causes the kernel to be rebuilt and the system to be rebooted.

Selecting the installation of the NFS component makes it possible to use RT work stations as NFS servers and clients. During installation, the NFS software components are added to the appropriate kernel libraries, and the NFS file system structure is defined in the kernel along with the instructions required for NFS applications.

NFS is configured into the kernel as a new virtual file system in the generic file system structure. The system defines NFS in the **sysconfig** file of the **/usr/sys** directory. It also adds the **libnfs** library of kernel code that links the NFS client and server code to the kernel and the NFS system calls to the same directory. An entry for NFS is added to the **/etc/vfs** file. The system checks the **/etc/vfs** file for information about the virtual file systems that are defined in the kernel. The **/etc/vfs** file is discussed in greater detail later in this chapter.

Selecting the installation of Yellow Pages makes it possible to consolidate system information onto a centralized data base and use the Yellow Pages to administer the information from the data base.

Selecting the installation of both the NFS and the YP components at the same time causes the components to be added to the appropriate places, and the kernel to be rebuilt.

A Note about Reinstalling NFS

If you reinstall NFS on top of itself, the system backs up the following files that you may have altered:

- **/etc/rc.include**
- **/etc/rc.tcpip**
- **/etc/rc.nfs**
- **/etc/inetd.conf**

The **/usr/lpp/nfs** directory serves as the root for these backups.

The NFS install program then reconfigures the real files in exactly the same manner as if it were installing NFS for the first time.

Configuring NFS on Your System

This section discusses how to configure an NFS network the first time you install it.

Note: Only users with superuser authority can configure NFS.

Planning Your NFS Implementation

Before you begin, you must plan your NFS implementation. Consider which nodes or work stations should be servers, and which ones should be set up as clients only. NFS servers ***export*** file systems to NFS clients. To ***export*** a file system means to make it available for use by other machines (clients) on the network. Access to exported file systems can be restricted to specified clients. Remember that a work station can be both a server and a client.

Configuring NFS the First Time

When you have decided on the NFS configuration of your network, do the following on all network nodes or work stations that are to be set up as NFS servers and clients:

1. Add the **/usr/etc** directory to the **PATH** variable.
2. Edit the **/etc/rc.include** file to start the **/etc/rc.tcpip** and **/etc/rc.nfs** programs.
3. Edit the **/etc/rc.tcpip** file to start the **portmap** and **inetd** daemon programs.
4. Customize each work station designated as an NFS server as follows:
   a. Edit the **/etc/inetd.conf** file to start the **mountd** daemon and any other RPC daemons you plan to run for your NFS implementation. See "NFS System Calls, Utilities, and Commands" on page 12-10 for the list of RPC daemons available.
   b. Edit the **/etc/rc.nfs** file to start the **nfsd** and **biod** daemons.
   c. Create the **/etc/exports** file.
5. Customize each work station designated as an NFS client as follows:
   a. Establish default NFS mounts if appropriate.
   b. Edit the **rc.nfs** file to start the **biod** daemon and to include a **mount** command that automatically mounts remote files when the system starts.
   c. If you plan to run RPC daemons, edit the **inetd.conf** file to start the daemons. See "NFS System Calls, Utilities, and Commands" on page 12-10 for the list of RPC daemons available.
6. If you are using personal computers in your network that are running PC-NFS, configure the **pcnfsd** program on the appropriate work stations.
7. Shut down the system and reboot to start NFS.

Note: If your network contains a variety of machines from different manufacturers, confirm that the maximum data packet size is consistent among the configured machines. Search the **/etc/net** file for the network device you are using for NFS and adjust the `inetlen` and `r_inetlen` entries.

Modifying the PATH Variable

The **/usr/etc** directory must be added to the **PATH** variable for system users. The **PATH** variable contains a list of the directory names that identify to the startup shell and other commands the locations of the executable files (programs or commands) for the system. Using the method described in "Tailoring the User Environment" on page 2-32, find the standard **PATH** variable for users and modify it to contain the **/usr/etc** directory.

Editing /etc/rc.include on All NFS Servers and Clients

The **/etc/rc.include** file contains commands that the system reads and implements when it starts, including commands to start the TCP/IP and NFS programs. The TCP/IP program must be properly configured on all NFS servers and clients. See *Interface Program for use with TCP/IP* for information on TCP/IP.

Starting the TCP/IP Interface Program

Search the **/etc/rc.include** file for the following entry:

```
sh /etc/rc.tcpip
```

If there is a comment (#) symbol at the beginning of the entry, delete the #. This directs the system to read the file containing startup instructions for TCP/IP.

Starting the NFS Program

There is also an entry in **/etc/rc.include** for the NFS program. Search the file for the following:

```
echo INIT Network File System
sh /etc/rc.nfs
```

If there are comment (#) symbols at the beginning of each line of the entry, delete the # at the beginning of each line. Uncommenting the lines directs the system to read the file containing startup instructions for NFS the next time the system starts.

Editing /etc/rc.tcpip On All NFS Servers and Clients

The **/etc/rc.tcpip** file contains entries for the **portmap** and **inetd** daemons, which are required for NFS.

Starting the portmap Daemon

The **portmap** daemon keeps track of which RPC services each RPC server handles and the ports the servers are listening on. Search the **/etc/rc.tcpip** file for the following entry:

```
if [ -f /etc/portmap ] ; then
   /etc/portmap ; echo ' portmap\c'    >/dev/console
fi
```

If comment (#) symbols appear at the beginning of each line in the entry, delete the # from the beginning of each line. Uncommenting the lines causes the **portmap** daemon, sometimes referred to as the **portmapper**, to start the next time **rc.tcpip** is run.

Starting the inetd Daemon

The **inetd** daemon is the internet server process. Search the **/etc/rc.tcpip** file for its entry:

```
if [ -f /etc/inetd ] ; then
   /etc/inetd ; echo ' inetd\c'    >/dev/console
fi
```

If comment (#) symbols appear at the beginning of each line in the entry, delete the # from the beginning of each line. Uncommenting the lines causes the **inetd** daemon to start the next time **rc.tcpip** is run.

Note: The **portmap** entry must appear before the **inetd** entry in the **etc/rc.tcpip** file. Make note of this if you must manually enter the lines for any reason.

Customizing an NFS Server

Customizing each work station designated as an NFS server involves editing the **/etc/inetd.conf** and **/etc/rc.nfs** files, and creating the **/etc/exports** file.

Editing /etc/inetd.conf

The **mountd** daemon and other daemons required for remote communication are started from the **/etc/inetd.conf** file.

Starting the mountd Daemon

The **mountd** daemon is called by the **mount** command to verify that the file system can be exported by the requesting client. In order for the remote mount to succeed, **mountd** must be active on the server. Search **/etc/inetd.conf** for the following **mountd** entry:

```
mountd sunrpc_udp udp wait root /usr/etc/rpc.mountd mountd 100005 1
```

If a comment (#) symbol appears at the beginning of this line, delete the #. Uncommenting the line causes the **mountd** daemon to start the next time **/etc/inetd** is run.

Note: If the **inetd** daemon is running on your network and it is possible that **mountd** is already active, you can enter the following at the command line to see if **mountd** is active:

`rpcinfo -u` *localhost* `100005`

Replace the *localhost* parameter with the host name of the server you are customizing. If you receive a message that the program is `ready and waiting`, **mountd** is already uncommented.

Starting Other RPC Daemons

In addition, search the file for any other RPC daemons you plan to activate. RPC daemons are identified by lines that looks similar to the **mountd** entry and contain the `sunrpc_udp` or `sunrpc_tcp` fields. If comment (#) symbols appear before the entries you plan to activate, remove the # from the beginning of each entry. See "NFS System Calls, Utilities, and Commands" on page 12-10 for information on these daemons.

Editing /etc/rc.nfs

The network file system daemon (**nfsd**) and block I/O daemon (**biod**) programs are started from the **/etc/rc.nfs** file.

Starting the nfsd Processes

The network file system daemon (**nfsd**) processes carry remote procedure call requests to the kernel for processing. Search the **/etc/rc.nfs** file for the following lines:

```
if [ -f /etc/nfsd -a -f /etc/exports ]; then
   /etc/nfsd 4 ; echo ' nfsd\c' >/dev/console
fi
```

The number in the second line (*4* in the example) determines the number of **nfsd** processes that are active. The load on the server determines the number of processes to assign. For an average load, four **nfsd** processes should be available to execute remote procedure calls.

If these lines do not appear, add them to the file after the following entry:

```
echo 'starting rpc and nfs services:\c'   > /dev/console
```

Starting the biod Processes

The block I/O daemon (**biod**) processes handle data transmission between NFS servers and clients. Search the **/etc/rc.nfs** file for the following lines:

```
if [ -f /etc/biod ]; then
   /etc/biod 4 ; echo ' biod\c'    > /dev/console
fi
```

If these lines do not appear, add them to the file following the `nfsd` entry.

Note: Like the `nfsd` entry, the number in the second line determines the number of **biod** processes that are started. The load on the server determines the number of processes to assign. For an average load, four **biod** processes should be available to handle NFS server and client data transmissions.

Creating the /etc/exports File

The **/etc/exports** file specifies the file systems that are available for mounting remotely. A server can only export its own file systems. A server cannot act as a messenger between clients and the file systems of other servers.

File systems can be made available to all clients, or client access to each file system can be controlled by listing the host names or authorized network groups (identified in the **/etc/netgroup** file) that are allowed to remotely mount each file system in the **/etc/exports** file. The **/etc/netgroup** file consists of lists that define network groups in specific patterns. See *AIX Operating System Technical Reference* for more information on **netgroup**.

Using a text editor, create the **/etc/exports** file. List by full path name each file system that can be exported. Align each file system name at the left margin. The path names must be the mount point of a local file system.

For file systems that can be mounted by all clients in the network, do not type any names following the file systems. If you are controlling client access, type the host names or the **netgroup** names following the name of the file system leaving white space between each name. If you need more than one line to list the names, start each new line with white space.

Note: You can add comments anywhere within the file by inserting the comment (#) symbol to signal the start of your comment text. The comment text extends to the end of the line on which the symbol appears.

The following shows examples of entries that can be created for an **/etc/exports** file:

```
/u                                  # export to the world
/tmp        mach1 mach2 mach3       # export to only these machines
/usr        clients                 # export to netgroup called clients
```

In this example, the first entry specifies that all clients can mount the `/u` directory. The second entry specifies that only the machines named `mach1`, `mach2`, and `mach3` can mount the `/tmp` directory. The third entry specifies that only the machines found in the **netgroup** called `clients` can mount the `/usr` file system.

Customizing an NFS Client

Customizing each work station designated as an NFS client involves editing the **/etc/rc.nfs** file to start the **biod** processes, and establishing default NFS mounts. In addition, you can edit the **/etc/inetd.conf** file to start any RPC daemons that you plan to run on your NFS client work station.

Establishing the Default NFS Mounts

Default NFS mounts refer to the remote file systems that are mounted automatically when the system starts. To establish the default NFS environment for clients, list the remote file systems that should be mounted automatically when the system starts in the **/etc/filesystems** file on each client. After you have set up the default NFS environment, the mounts are made automatically when the system starts. The specified file systems do not have to be mounted from the command line each time the system reboots. However, other file systems can be mounted remotely from the command line if the appropriate conditions exist for the client. See "Mounting Files Remotely From the Command Line" on page 12-32 for detailed information.

Editing /etc/filesystems

The remote file systems that NFS clients attempt to automatically mount when the system starts are identified in the **/etc/filesystems** file. The **/etc/filesystems** file is organized in stanzas. Stanzas are lines of attributes grouped together by common functions. In the case of NFS mounts, the first line of each stanza should be the mount point of the file system being mounted. The mount point is followed by a colon (:). The attributes associated with the mount are listed on subsequent lines. See the file formats section in *AIX Operating System Technical Reference* for more detailed information on the **filesystems** file.

Using a text editor, add stanzas to **/etc/filesystems** to define each NFS mount. The following attributes are required for the stanzas you create that pertain to the NFS mounts:

dev = *filesystem_name*
For NFS mounts, *filesystem_name* specifies the path name of the remote file system being mounted.

mount = false
NFS mounts must use the `false` mount attribute.

nodename = *hostname*
Determines the host machine on which the remote file system resides.

vfs = nfs
Specifies that the virtual file system being mounted is an NFS file system.

In addition to these required attributes, you can use the following in the stanzas for the NFS mounts:

type = *type_name*
Defines the file system being mounted as part of the *type_name* mount group. This parameter is used in conjunction with `mount -t`, which mounts groups of specified file systems at the same time.

options=*options*
Specifies one or more of the following options:

bg	If the first mount attempt fails, try the mount again in the background.
fg	Try the mount again in the foreground.
retry=*n*	Set the number of times to try the mount.
rsize=*n*	Set the **read** buffer size to the number of bytes specified by *n*.
wsize=*n*	Set the **write** buffer size to the number of bytes specified by *n*.
timeo=*n*	Set the NFS timeout to the tenths of a second specified by *n*. Use this option to avoid situations that can occur in networks where the server load can cause inadequate response time.
retrans=*n*	Set the number of NFS retransmissions to the number specified by *n*.
port=*n*	Set the server Internet address port to the number specified by *n*.
soft	Return an error if the server does not respond.
hard	Continue to try the request until the server responds.

intr Allow keyboard interrupts on hard mounts.

ro Read-only.

rw Read-write. It is recommended to use the **hard** option along with this option to avoid error conditions that can conflict with applications if a **soft** mount is attempted as **read-write**.

The **bg** and **fg** options cause the **mount** command to run in the background (**bg**) or foreground (**fg**) if the server's **mountd** process does not respond. The system attempts to transmit each request the number of times specified in the **retry** option before it gives up. Once the file system is mounted, each NFS mount request made in the kernel waits for a response during the time specified in the **timeout** option. If no response arrives, the time-out is multiplied by two, and the request is transmitted again. When the number of retransmitted attempts for a mount request that specifies the **soft** option have been sent with no reply, an error is returned. When the number of retransmitted attempts for a mount request that specifies the **hard** option have been sent with no reply, the system displays a message and retries the request.

When you specify a **hard** mount, it is possible that the process can hang while waiting for a response. To be able to interrupt the process and end it from the keyboard, use the **intr** option in the mount options. The number of bytes in a **read** or **write** request can be set with the **rsize** and **wsize** options. If you do not set these options, the kernel automatically sets them.

The system uses the following as defaults:

```
fg
retry=10000
timeo=7
retrans=3
port=NFS_PORT
hard
rsize=8192
wsize=8192
```

This means that the system runs the NFS mount request in the foreground. Once the file system is mounted, each NFS mount request made in the kernel waits seven tenths (0.7) of a second for a response. If no response arrives, the time-out is multiplied by 2 and the request is retransmitted. When the request has been retransmitted three times with no reply, the file system prints a message and retries the request. The port is set to the value of **NFS_PORT** as defined as a constant in the kernel. The kernel sets the **rsize** and **wsize** options (to 8192 bytes). See *AIX Operating System Commands Reference* for more detailed information on the **mount** command.

The following is an example of an NFS stanza that could appear in **/etc/filesystems**. In this example, you are setting up the NFS client known as `mach1` and have already established the mount points for the nodes in your network. The following stanza can be created for an automatic remote mount:

```
/u/jdoe:
    dev = /u/jdoe
    mount = false
    vfs = nfs
    nodename = mach2
    options = ro,soft
    type = nfs_mount
```

This stanza directs the system to mount the remote directory `/u/jdoe` over the local mount point of the same name. The file system is mounted as read-only, and since it is mounted as **soft**, an error is returned in the event the server does not respond. By specifying the **type** as `nfs_mount`, the system attempts to mount `/u/jdoe` (along with any other file systems that are specified in the `type = nfs_mount` group) when the command `mount -t nfs_mount` is issued.

Editing /etc/rc.nfs

The **biod** processes are started from the **/etc/rc.nfs** file. In addition, NFS file system mounts can be made automatically when the system starts if **/etc/rc.nfs** contains a mount command for remote file systems.

Starting the biod Daemons

The block I/O daemon (**biod**) processes handle data transmission between the NFS clients and servers. Search the **/etc/rc.nfs** file for the following lines:

```
if [ -f /etc/biod ]; then
   /etc/biod 4 ; echo ' biod\c'    > /dev/console
fi
```

If these lines do not appear, add them to the file after the following entry:

```
echo 'starting rpc and nfs services:\c'   > /dev/console
```

Note: The number in the second line of each entry determines the number of **biod** processes that are active. The load on the server determines the number of processes to assign. For an average load, four **biod** processes should be available to handle NFS client and server data transmissions.

Adding a mount Command Sequence for Automatic Remote Mounts

To mount NFS file systems automatically whenever the system is restarted, edit the **/etc/rc.nfs** file to include a mount command for remote file systems.

Note: File systems that are mounted automatically must be defined in the **/etc/filesystems** file discussed in "Editing /etc/filesystems" on page 12-22.

Search the **/etc/rc.nfs** file for the following lines:

```
mount -v nfs all
mount -t xxx
```

These lines are commented out by default. Uncomment one of these lines by removing the comment (#) from the beginning of the entry. Uncomment only one line to enable NFS mounts whenever the **/etc/rc.nfs** file is run.

If you uncomment the line `mount -v nfs all`, the system attempts to mount every file system defined in **/etc/filesystems** that contains the stanza attribute `vfs = nfs`.

To mount selected NFS file systems, uncomment the line `mount -t xxx`. Replace `xxx` with the *type_name* you define in **/etc/filesystems**. When the **/etc/rc.nfs** file is executed, the system attempts to mount every file system that contains the stanza attribute `type =` *type_name*.

Establishing the Local Mount Points

Confirm that the mount points for NFS mounts you defined in **/etc/filesystems** exist. If they do not exist, you need to create them. Using the **mkdir** (make directory) command, create the mount points in the following form:

`mkdir` *dirname*

Editing the /etc/vfs File

While NFS is being installed, the system adds an entry to the **/etc/vfs** file for the virtual file system that will be known as an NFS file system. To make NFS the default remote file system, add the following line to **/etc/vfs**:

```
%defaultvfs aix nfs
```

Editing /etc/inetd.conf

If you plan to use any of the RPC daemon programs to assist in remote communications between the client and other network machines, you must edit the **/etc/inetd.conf** file to activate the appropriate daemons. The **/etc/inetd.conf** file is shipped with the RPC daemon entries in place, but some of the RPC daemons may be commented out by default. In order to activate the RPC daemons you plan to use, search the file for the existing entries. The RPC daemon entries can be identified by a field listed as `sunrpc_udp` or `sunrpc_tcp`. Remove the comment (#) symbols that appear before the entries to activate the daemons. See "NFS System Calls, Utilities, and Commands" on page 12-10 for the list of daemons available and *AIX Operating System Commands Reference* for information on each daemon.

If the **inetd** process is already running on the work station and you have made changes to the **/etc/inetd.conf** file, send a SIGINT (signal interrupt) to the **inetd** process. SIGINT causes the **inetd** process to reread the **inetd.conf** file so the changes you have made can be implemented. To issue SIGINT, type the following at the command line:

`kill -2` *pid*

The *pid* is the process ID number for the current **inetd** process that is running.

Configuring pcnfsd

If your network includes personal computers that are running versions of the Disk Operating System (DOS) and have the PC-NFS[3] program installed, you can configure the **pcnfsd** daemon on network servers to allow users on the PC-NFS clients to access authentication and printer spooling services. Without the **pcnfsd** program, users on PC-NFS client systems can perform many PC-NFS functions that do not involve specialized server support, such as mounting exported file systems and requesting network addresses and host names, but cannot access authentication and printer spooling services.

The authentication capabilities of the **pcnfsd** program enable the person who manages the system to monitor system resources, and to select the appropriate system security. Authentication is a way of recognizing individual users and assigning them different privileges. In the typical PC-NFS environment, any user on a personal computer that is running PC-NFS can access NFS servers, with **nobody** privileges. With **nobody** privileges, there is no way to tell which user has accessed a resource because all personal computer user files show up as owned by `nobody`. Universal use of **nobody** privileges makes it difficult for the person who manages the system to account for use of system disk space. When **pcnfsd** is installed, a PC-NFS user can issue the **net name** command from a personal computer to log in to PC-NFS in the same manner as the user can log in to AIX. The user's name and password are verified by **pcnfsd**. This authentication procedure does not make a server more secure, but it does give a user more control over access to files that are available through NFS.

When it starts, **pcnfsd** invokes the standard print spooling subsystem, which allows it to be used with remote printers. When properly configured, PC-NFS redirects files intended for personal computer printers to a network print spooling service. The **pcnfsd** process places the personal computer file in a spooling directory, and invokes the printing function with appropriate parameters. The directory used for spooling is passed as an argument to **pcnfsd** when it is started. The directory must be in an exported file system that is accessible to PC-NFS clients. When PC-NFS determines that a file is to be printed, PC-NFS mounts the spool directory as an NFS virtual drive, and creates spool files in the directory. PC-NFS calls **pcnfsd** with the following information:

- The name of the file to be printed
- The login ID of the user on the client
- The name of the printer to be used.

The **pcnfsd** daemon then invokes the local spooler to print the file.

---

3 PC-NFS is a product of Sun Microsystems, Inc. The PC-NFS program allows personal computers running DOS as their operating system to be configured into the Network File System running on a network that has computers running various other operating systems, including AIX Operating System.

The **pcnfsd** daemon program is installed with the NFS component of IBM AIX/RT Network File System licensed program. You should configure **pcnfsd** on the following:

- Systems that perform user authentication services
- Systems that offer print spooling
- All Yellow Pages servers (YP master servers and YP slave servers).

  Note: Typical Yellow Pages (YP) networks are configured so that PC-NFS can pick any YP server as the default server, so it is important that all servers have **pcnfsd** running. If running **pcnfsd** on all YP servers is not practical, or you want to limit requests to a specific server, add a **net pcnfsd** command to the **autoexec.bat** file on each personal computer to force it to use a specific YP server.

Before configuring **pcnfsd**, select a suitable location for the spool directories such as the **/usr/tmp** directory. The spool directory must have at least 100 kbytes of free space. Next, export the file system containing the spool directories. Do not put access restrictions on the exported file system that could cause access problems in your network.

Starting pcnfsd

You can start **pcnfsd** in different ways, depending on the spooling directory you want to use.

Using the Default Spooling Directory

If you intend to use **/usr/tmp** as the default spooling directory you can start **pcnfsd** from the **inetd.conf** file. Search for the following entry in **inetd.conf** file:

```
pcnfsd sunrpc_udp udp wait root /etc/rpc.pcnfsd pcnfsd 150001 1
```

If a comment (#) symbol appears at the beginning of this entry, delete the #. The next time **inetd** is run, **pcnfsd** print spooling requests are sent to the print spooling directories in **/usr/tmp**.

Note: You cannot change the name of the default directory in the **inetd.conf**.

Specifying a Particular Spooling Directory

To use a directory other than the default spooling directory **/usr/tmp**, add the following entry to the **/etc/rc.nfs** file:

```
if [ -f /etc/rpc.pcnfsd ] ; then
        /etc/rpc.pcnfsd -s spooldir ; echo " rpc.pcnfsd\c"    >/dev/console
fi
```

The *spooldir* parameter specifies the name of the directory to be assigned as the **pcnfsd** print spooling directory. Start the **pcnfsd** print spooler by entering the following at the command line:

`/etc/rpc.pcnfsd -s` *spooldir*

Note: When you specify the print spooling directory, be sure to deactivate the **pcnfsd** print spooling directory default entry in the **inetd.conf** file. Search the file for the `pcnfsd sunrpc_udp` entry. If it is not preceded by a comment (#) symbol, add a # to the beginning of the line to suppress the entry.

Maintaining the Spool Directory

Since printer redirection requests cause file listings of zero length to be left in the PC-NFS spool directories, you should periodically clear spooling directories of these entries. You can create a shell script that removes the zero-length files, and have the **cron** process execute it at appropriate intervals. See *AIX Operating System Technical Reference* for detailed information on the **cron** process.

Verifying that pcnfsd is Accessible

If you have started **pcnfsd** at the command line, verify that **pcnfsd** is accessible through the network by entering the following at the command line:

`rpcinfo -u` *localhost* `150001 1`

The *localhost* parameter specifies the host name of the machine on which you are configuring **pcnfsd**. After you have entered the command, you should receive the message that the program is `ready and waiting`.

See *AIX Operating System Commands Reference* for detailed information on the **pcnfsd** command.

Shutting Down and Rebooting Each System

After you have finished customizing the NFS server and client work stations, shut down and reboot each system by entering the following at the command line:

```
shutdown -r
```

When the system comes back up, the NFS environment you have configured is available.

Note: If the Yellow Pages network service is already configured in your network, the order in which you reboot the machines is important. It is recommended that you reboot the YP master server first, followed by the YP slave servers and the YP clients. This is explained in greater detail in "Configuring the Yellow Pages on Your System" on page 12-38.

Maintaining NFS

Once you have configured NFS on your system, you can make changes to the configuration without going through the entire NFS installation procedure. This section discusses maintenance tasks, such as adding network daemons, mounting individual file systems from the command line, and removing the superuser restrictions on mounted file systems.

Changing the Number of Network Daemons

You can change the number of active **nfsd** processes on NFS servers, and the active number of **biod** processes on both NFS servers and clients, at the command line. Use the following format:

/etc/*daemon* #

Replace the *daemon* parameter with **nfsd** or **biod**, and the # with the number of processes you want to have active. When you press **Enter**, the new number of daemon processes are available. You do not have to reboot the system to activate the daemons. For example, to increase the number of **biod** daemons currently active from two to five, enter the following:

```
/etc/biod 3
```

When you press **Enter**, there are five **biod** daemons available.

Changing the inetd.conf file

If the **inetd** process is already running on the work station and you have made changes to the **/etc/inetd.conf** file, send a SIGINT (signal interrupt) to the **inetd** process. SIGINT causes the **inetd** process to reread the **inetd.conf** file so the changes you have made can be implemented. To issue SIGINT, type the following at the command line:

kill -2 *pid*

The *pid* is the process ID number for the current **inetd** process that is running.

Mounting Files Remotely From the Command Line

A remote file system that is not mounted automatically can be mounted at an NFS client from the command line using the **mount** command. The file system containing the directory to be mounted must be listed in the **/etc/exports** file on the server from which it is being mounted, and the local mount point must exist. In addition, the user must be authorized to use the **mount** command. (See *AIX Operating System Commands Reference* for details on using the mount command.)

Mounting all NFS File Systems Defined in /etc/filesystems

To mount all NFS mounts defined in **/etc/filesystems**, enter the following at the command line:

```
mount -v nfs all
```

To mount a specific NFS file system defined in **/etc/filesystems**, enter the following:

`mount` *dirname*

The *dirname* parameter is the mount point (full path name) of the file system being mounted. For example, a file called `/reports/weekl` can be mounted from the command line if the stanza for the file exists in in the **/etc/filesystems** file, and the file is listed in the server's **/etc/exports** file with no restrictions. To mount the file system, enter the following:

```
mount /reports/weekl
```

Mounting Selected NFS File Systems Defined in /etc/filesystems

If there are groups of multiple remote file systems that you would like to access at one time, you can define each of the files in a group with the same *type_name* in their stanza in **/etc/filesystems**. To mount selected NFS file systems as a group, enter the following:

`mount -t` *type_name*

The *type_name* parameter refers to the unique name you have defined to represent a group of file systems.

For example, the file systems maintained by a special department, such as the engineering department, could be defined with the **type** attribute in the appropriate **/etc/filesystems** stanzas as follows:

```
type = engineering
```

Then, using the following command, you can instruct the system to mount all the file systems with the `engineering` type attribute:

```
mount -t engineering
```

For file systems that belong to multiple groups and require mounting on a daily basis, you can add `mount -t` entries for those mount groups to the end of **/etc/rc.nfs**. This causes the system to attempt to mount the file systems associated with the groups you had defined whenever the system is restarted.

Mounting NFS File Systems Not Defined in /etc/filesystems

To mount a remote NFS file system that is not defined in /etc/filesystems, enter the following at the command line:

`mount -n` *hostname* `-v nfs [-o` *options*`]` *dirname1 dirname2*

The *hostname* parameter identifies which server the file system is being mounted from. The *options* parameter specifies the options available to NFS mounts, such as read-only (**ro**), read-write (**rw**), hard, or soft. The *dirname1* parameter identifies the name of the remote file system being mounted, while *dirname2* identifies the name of the local directory *dirname1* is being mounted on (the mount point).

For example, the `/reports/week1` file system exists on the server called `mach1` and is listed in **/etc/exports** with no restrictions. You want to mount the file system on the client `mach2` as read-only and as a soft mount. To mount the file system on your local directory `/summary/reports`, enter the following:

```
mount -n mach1 -v nfs -o ro,soft /reports/week1 /summary/reports
```

Note: If the default virtual file system for the remote file system being mounted is already defined as NFS in the **/etc/vfs** file, you can omit the `-v nfs` flag from the above sequence. Your entry at the command line would appear as follows:

`mount -n` *hostname* `[-o` *options*`]` *dirname1 dirname2*

See the preceding example for the parameter descriptions.

Removing Superuser Privileges on Mounted File Systems

To prevent unauthorized access to NFS servers, NFS does not allow a user on a client to exercise superuser privileges on files in a mounted file system. The root user ID on the client (0) is mapped to the kernel variable **nobody** (−2, or unsigned 65534) when performing file operations in a mounted file system.

Restrictions on Superuser from a Client

The following examples demonstrate restricted privileges for the superuser on an NFS mounted file system. First, as a non-privileged user from a client, create two files in a mounted file system on which the user has write permission. Modify the permissions on each file as shown:

```
cd mounted_filesystem
touch demo1
touch demo2
chmod 700 demo1
chmod 777 demo2
```

Next, become the superuser on the client and attempt to access the files you created as an ordinary user by entering the following at the command line:

```
touch demo1
```

Although you are logged in as the superuser, you are not permitted to modify this file. File access is prohibited because the user ID `0` has been mapped to `65534` for file operations on the mounted file system. No one but root on the host exporting the file system and the file's owner can modify the file.

In the case of `demo2`, the superuser on the client can access this file because the **write** permission for `others` is set.

Programs that run as root (**setuid root** programs) have similar restrictions. A user running a **setuid root** program from a client is not allowed to do privileged operations on files in NFS mounted file systems. Also note that changing ownership of a file in a mounted file system to `root` means that it will be owned by root on the server, not root on the client.

Removing the Superuser Restriction

The superuser restriction can be removed by altering the value of the **nobody** kernel variable on the NFS server. Any or all of the following can be altered:

- The copy of the kernel in memory
- The binary image of the kernel (**/unix**)
- The object file from which new kernels are built.

If you alter only the copy of the kernel in memory, the change will be lost the next time you restart the system. If you alter the binary image, the change will be preserved until the next time the kernel is rebuilt. If you alter the object file, any new kernels you build will include the change.

Warning: Removing the superuser restrictions on an NFS server means that any client superuser will also have superuser privileges on that host. It is not recommended to make this modification, especially if one of the following is true:

- Access to root is not tightly controlled on all machines in the network.
- All machines in the network are not administered by a single person or staff.

Altering the Kernel in Memory

To change the kernel in memory on a running system, do the following:

1. On the NFS server, change the value of the **nobody** kernel variable by using the **adb** command. Enter the following at the command line:

   ```
   adb -w /unix /dev/kmem
   ```

   The system may respond with a message that looks like the following:

   ```
   bad core magic number
   ```

2. To check the current setting of the **nobody** variable, enter the following at the command line:

   ```
   /m 0 -1 0
   nobody/D
   ```

   The following message should be displayed by the system:

   ```
   _nobody:    -2
   ```

3. To reset the **nobody** variable to zero, enter the following at the command line:

   ```
   nobody/W0
   ```

   The following message should be displayed by the system:

   ```
   _nobody:    -2  =      0
   ```

   Exit **adb** by pressing **Ctrl-D**.

Note: To change the **nobody** variable back to -2, you can do one of the following:

- Use the same procedure as you used to alter the kernel, but replace `nobody/W-2` with `nobody/W0`, or
- If you haven't modified the binary image, restart the system.

Altering the Binary Image

To change the binary image of the kernel (**/unix**), do the following:

1. Save a copy of the unmodified kernel by entering the following at the command line:

   ```
   cp /unix /unix.SAFE
   ```

2. To check the current setting for the **nobody** variable, enter the following:

   ```
   adb -w /unix
   nobody?
   ```

   The following message is displayed:

   ```
   _nobody:    -2
   ```

3. Reset **nobody** to 0 by entering the following:

   ```
   nobody?W0
   ```

   The following message is displayed:

   ```
   96ac0:    -2  =    0
   ```

4. Exit **adb** by pressing **Ctrl-D**.

If you modify **/unix** without also modifying the copy in memory, you will have to reboot for the change to take effect.

Note: To change the **nobody** variable in **/unix** back to -2, you can do one of the following:

- Use the same procedure that you used to alter the binary image, but replace `nobody?W0` with `nobody?W-2`, or
- If you saved a copy of the old kernel, enter the following at the command line:

  ```
  mv /unix.SAFE /unix
  ```

Altering the Object File

To change the object file that is used to build new kernels, do the following:

1. Change to the **/usr/sys** directory, extract the object file from the **libnfs** archive, and save a copy by entering the following at the command line:

   ```
   cd /usr/sys
   ar x libnfs nfs_server.o
   cp nfs_server.o nfs_server.o.S
   ```

2. Check the current setting for the **nobody** variable by entering the following:

```
adb -w nfs_server.o
nobody?D
```

The following message is displayed:

```
_nobody:    -2
```

3. Reset **nobody** to 0 by entering the following:

```
nobody?W0
```

The following message is displayed:

```
276c:    -2  =    0
```

4. Exit **adb** by pressing **Ctrl-D**.

5. Place the modified object file in the **libnfs** archive and remove the file from the current directory by entering the following:

```
ar rv libnfs nfs_server.o
rm nfs_server.o
```

The next time you build a new kernel, the modification is incorporated.

Note: To change the **nobody** variable in the object file back to -2, you can do one of the following:

- Use the same procedure you used to alter the object file, but replace `nobody?W-2` with `nobody?W0`, or
- If you still have the copy of the old object file you saved, enter the following at the command line:

```
mv nfs_server.o.S nfs_server.o
ar rv libnfs nfs_server.o
```

This places the original version of the object file back into **libnfs**.

Configuring the Yellow Pages on Your System

This section discusses how to configure the YP for the first time in your network.

Note: Only users with superuser authority can configure and modify Yellow Pages.

Planning Your YP Implementation

Before you begin, you must plan your Yellow Pages implementation. Consider which nodes or hosts are going to use YP. When you have determined the hosts that will be using YP, assign them to YP domains. A YP domain refers collectively to a group of a hosts. In many cases, a single domain is sufficient for an NFS network.

In each YP domain, system and user account information is consolidated in a set of data bases, also called ***YP maps***. It is recommended that you create and maintain all YP maps for one YP domain on a single host. The host on which you maintain a YP map is called the YP master server. One or more hosts should be designated to maintain exact replicas of the YP maps on the YP master server. These hosts are called YP slave servers. Use at least one YP slave server per domain to help balance the YP processing task load, and to provide YP services in the event the YP master server is not available. The remaining hosts should be designated as YP clients. YP clients do not maintain any YP maps. They query the YP servers for system and user account information. The YP clients do not have to make a distinction between querying the YP master server or a YP slave server.

Configuring Yellow Pages

When you have decided on the YP configuration of your network, do the following:

1. On all YP hosts, (the YP master server, YP slave servers, and YP clients), add the **/etc/yp** directory to the **PATH** variable.
2. On all YP hosts (the YP master server, YP slave servers, and YP clients), edit the **/etc/rc.nfs** file as follows:
   a. Change the default name of the domain to the one you have selected.
   b. Start the **ypbind** process.
3. On each YP host, set the domain with the **domainname** command.
4. Customize the YP master server as follows:
   a. Edit the **/etc/rc.nfs** file to start the **ypserv** process.
   b. Edit the input files that are used to create YP maps.
   c. Create the YP maps with the `ypinit -m` command.
   d. If the users in your YP network can change their own passwords, edit the **/etc/rc.nfs** file to start the **yppasswdd** daemon.
5. Customize each YP slave server as follows:
   a. Edit the **/etc/rc.nfs** file to start the **ypserv** process.
   b. Edit selected system files to direct requests to the YP service.
   c. Replicate the YP maps from the YP master server with the `ypinit -s` command.
6. Customize each YP client by editing selected system files to direct requests to the YP service.

Modifying the PATH Variable

The **/etc/yp** directory must be added to the **PATH** variable for system users. The **PATH** variable contains a list of the directory names that identify to the startup shell and other commands the locations of the executable files (programs or commands) for the system. Using the method described in "Tailoring the User Environment" on page 2-32, find the standard **PATH** variable for users and modify it to contain the **/etc/yp** directory.

Editing /etc/rc.nfs on All YP Hosts

The **/etc/rc.nfs** file contains NFS and YP commands that the system implements when it starts. To customize your YP environment, you must modify the default domain name and the **ypbind** entries that already exist in **/etc/rc.nfs**.

Changing the Default Name of the Domain

Search the file for the following entry:

`/bin/domainname` *domain*

The *domain* parameter is often specified as `ibm` by default in this file. Replace *domain* (or `ibm`) with the name of the YP domain in which the YP host resides. The next time **/etc/rc.nfs** runs, the domain is set to the name you supply.

Starting ypbind

The **ypbind** process searches for a YP host that has YP maps to match a YP client's request. Search the file for the following entry:

```
if [ -f /etc/ypbind ] ; then
   /etc/ypbind ; echo ‘ ypbind\c‘   >/dev/console
fi
```

In most cases, these lines are commented out by default. If the lines are preceded by a comment (#) symbol, remove the # that precedes each line. The next time **/etc/rc.nfs** runs, the **ypbind** process starts.

Setting the YP Domain

The YP domain must be set on each YP host before the YP maps can be created and distributed. Set the domain by entering the following at the command line:

`domainname` *domain*

Replace the *domain* parameter with the same one you used in the **/etc/rc.nfs** file.

Customizing the YP Master Server

Customizing the YP master server involves editing the input files used for YP maps, creating the YP maps to generate a master YP data base, and starting the YP processes that consult the maps upon YP client request. In addition, if the users in your YP network can change their own passwords, you must also edit the **/etc/rc.nfs** file to start the **yppasswdd** daemon.

Editing the YP Map Input Files

The YP maps are usually constructed from text files that contain information in a standardized format. The AIX files used to make the default YP maps include the following:

- **/etc/passwd** (or a specially designated YP password account file)
- **/etc/group** (or a specially designated YP group account file)
- **/etc/hosts**
- **/etc/rpc**
- **/etc/protocols**.
- **/etc/services**

In addition, default YP maps are created from the following files if they are available on the work station:

- **/etc/ethers**
- **/etc/netgroup**
- **/etc/networks**
- **/usr/lib/aliases**

It is important to create and maintain all information that may be required by any host in the YP domain on the YP master server. For example, if there are any special RPC program entries that are required by other hosts in the domain, the entries must be added to the **/etc/rpc** file on the YP master server, even if the YP master server does not use those programs.

After the YP maps are installed on the YP servers, system processes that ordinarily access these text files are redirected to the appropriate YP map for the information. The exception to this involves the **/etc/passwd**, **/etc/group**, and **/etc/hosts** files.

The **/etc/hosts** file is consulted by the system startup processes before the YP is activated, but subsequent queries for host information are redirected to a YP map. In order to provide selected user and group information to the entire domain while maintaining certain information for local use only, system processes that require user and group information first consult the **/etc/passwd** and **/etc/group** files on the local machine.

If the processes encounter a special YP escape sequence in the file before their query has been answered, they are redirected to the YP map. You can install user and group information in your YP environment in one of the following manners:

- Use **/etc/passwd** and **/etc/group** on the YP master as the YP map input files.
- Use separate password and group files as the YP map input files.
- Do not use YP for password and group information.

Determine which situation is appropriate for your YP network environment by considering your administrative overhead.

Using /etc/passwd and /etc/group as Map Input

By default, YP uses the **/etc/passwd** and **/etc/group** files from the YP master server as input files for the YP maps that contain password and group information.

All users and groups on the YP master server are included automatically in the YP maps. In a network that contains work stations from a variety of manufacturers, the user and group IDs supplied for a work station may conflict with those on the YP master server. This can cause file ownership problems if an NFS server exports a directory to an NFS client whose password file contains user IDs that match those in the YP map.

Note: In order for users listed in the YP password files to change their passwords, the **yppasswdd** must be running on the YP master server, and they must use the **yppasswd** command.

Warning: Do not put the YP escape entry in the files you use as the input password and group files for YP maps. This causes security problems that may affect all work stations on the network.

Using Separate Password and Group Files as YP Map Input

To permit exclusion of selected entries from the YP maps, you can create new files, such as **/etc/yp/passwd** and **/etc/yp/group**, that are for YP users and groups only. Place entries designated for local use only in **/etc/passwd** and **/etc/group**, and place entries designated for YP use in the new files. To configure the YP master server in this way, do the following:

1. In **/etc/passwd**, verify that existing entries are local entries only and add the YP escape sequence (`+::0:0::`) as the last line in the file.
2. In **/etc/group**, verify that existing entries are local entries only and add the YP escape sequence (`+:`) as the last line in the file.
3. Create new files for the YP password and group entries. These files should have the same format as **/etc/passwd** and **/etc/group**.

4. Modify the following variables in the **/etc/yp/Makefile** file:

   a. Change `PASSWD = /etc/passwd` to `PASSWD =` *yp_pwfile*
      The *yp_pwfile* parameter specifies the full path name of the file created especially for YP user accounts.

   b. Change `GROUP = /etc/group` to `GROUP =` *yp_grpfile*
      The *yp_grpfile* parameter specifies the full path name of the file created especially for YP group accounts.

Note: You cannot use the **adduser** login ID or the **users** command to add YP users and groups.

Not Using YP for Password and Group Information

In a typical NFS environment, home directories are exported over the network so that users can log in from any work station and access their own working environment. To ensure proper permissions for remotely mounting file systems and to allow access to home directories throughout the network, NFS requires global user ID (UID) and group ID (GID) assignments.

If you choose not to use YP to manage UID/GID assignments in an NFS environment, you must maintain duplicate password and group accounts for users who need to access multiple NFS hosts. For example, a user on a work station whose UID is 205 and GID is 35 must have the same UID and GID (205 and 35) in **/etc/passwd** on any other work station the user accesses.

Creating the YP Maps on the YP Master Server

After the text files on the YP master server have been edited, you can build the default YP maps. Change to the **/etc/yp** directory and begin the **ypinit** process on the YP master server by issuing the following at the command line:

```
cd /etc/yp
ypinit -m
```

The system displays a message about quitting the **ypinit** process if it encounters non-fatal errors along with the default answer `NO`. Press **Enter**.

Note: If the system reports errors during the **ypinit** process, run the `ypinit -m` command again after you have resolved the problems.

Next, the system prompts you for the list of hosts you have designated as YP servers (in addition to the YP master server). Enter the name of each host you plan to use as a YP slave server. When you are finished with the list, press **CTRL-D** to continue the **ypinit** process.

As the **ypinit** process continues, it creates a subdirectory in **/etc/yp** that corresponds to the name of the YP domain you have set. The **ypinit** process then uses the instructions in

/etc/yp/Makefile on the YP master server to create the default YP maps, which are placed in the new subdirectory identified by the YP domain name. Each YP map consists of two files called *map.key*.pag and *map.key*.dir. The *map* parameter is the name of the map, such as **passwd** or **group** . The *key* parameter specifies the index criterion, such as **nam** for indexing entries by name or **uid** for indexing entries by user ID. The **.pag** and **.dir** filename extensions are used by the **makedbm** procedure, which is explained later in this section.

Note: To resolve naming convention differences in the *map* and *key* parameters, the YP processes consult the **/etc/yp/YP_MAP_X_LATE** file for translated names that AIX allows. You can add entries that are specific to your site to this file. See *AIX Operating System Technical Reference* for more information.

Starting the ypserv and ypbind Processes

When the **ypinit** process finishes building the YP maps, start the YP daemons on the YP master server.

The **ypserv** process runs as a background daemon on both the YP master server and YP slave servers to answer YP client requests for information. To start **ypserv**, enter the following at the command line:

```
/usr/etc/ypserv
```

If the system can locate the YP map subdirectory created by the **ypinit** process (the /etc/yp/*domain* directory created in the previous section), **ypserv** starts automatically from the **/etc/rc.nfs** file the next time the system is booted.

The **ypbind** process runs as a background daemon on all YP hosts to find the YP server with a YP map that has the information requested by a client. To start the **ypbind** utility, enter the following at the command line:

```
/etc/ypbind
```

If the entry for **ypbind** is uncommented in the **/etc/rc.nfs** file, **ypbind** starts automatically the next time the system is booted.

Starting the yppasswdd Daemon

If you use the **/etc/passwd** file as the input file or use a separate file (**/etc/yp/passwd**) for your YP password map, it is important to start the **yppasswdd** daemon on the YP master server. Without the **yppasswdd** daemon, users may not be able to change their passwords. If the **yppasswdd** daemon is running on the YP master server, users can use the **yppasswd** command at any YP host in the domain to change their YP password. An entry for the **yppasswdd** daemon is included in the **/etc/rc.nfs** file, but is usually commented out by default.

To activate the **yppasswdd** daemon on the YP master server, search the **/etc/rc.nfs** file for the following:

```
PASSWD=/etc/passwd
if [ -x /usr/etc/rpc.yppasswdd -a -f $PASSWD ] ; then
    /usr/etc/rpc.yppasswdd $PASSWD -m passwd PASSWD=$PASSWD
    echo ' yppasswdd\c'  > /dev/console
fi
```

Remove the comment symbols (#) that precede these lines. The next time the system boots, the **yppasswdd** daemon will run.

Note: Pay close attention to the file listed in the **PASSWD** argument. You can replace the file name (**/etc/passwd** or **/etc/yp/passwd** usually appears) with the name of the file you have selected as the input file for the YP password maps.

See *AIX Operating System Commands Reference* for more information on the **yppasswd** command and the **yppasswdd** daemon.

Customizing YP Slave Servers

Customizing work stations to be YP slave servers involves starting the YP processes that consult the maps, editing the three local access files that are not replaced by YP maps (**/etc/passwd**, **/etc/group**, and **/etc/hosts**), and replicating the YP maps from the YP master server.

Starting the ypserv and ypbind Processes

The **ypserv** process runs as a background daemon on both the YP master server and YP slave servers to answer YP client requests for information. To start **ypserv**, enter the following at the command line:

```
/usr/etc/ypserv
```

If the system can locate the YP map subdirectory created by the **ypinit** process (the /etc/yp/*domain* directory, which is created in "Transferring Maps From the YP Master Server" on page 12-47), **ypserv** starts automatically from the **/etc/rc.nfs** file the next the system is booted.

The **ypbind** process runs as a background daemon on all YP hosts to find the YP server with a YP map that has the information requested by a client. To start the **ypbind** utility, enter the following at the command line:

```
/etc/ypbind
```

If the entry for **ypbind** is uncommented in the **/etc/rc.nfs** file, **ypbind** starts automatically the next time the system is booted.

Editing the Local Access Files

In most cases, system processes are automatically redirected to YP maps for information. For example, a process looking up the name of a network service checks the YP **services.byname** (or **srvcs.nam**) map instead of the **/etc/services** text file on the work station when the YP is running. The work station text files that are replaced by YP maps are maintained on the YP master server only.

The following files, however, may contain information for local use only:

- **/etc/passwd**
- **/etc/group**
- **/etc/hosts**

The **/etc/passwd** and **/etc/group** files are augmented rather than replaced by YP maps. To maintain private information and still obtain information from the YP maps, special YP escape entries can be added to the files. The system processes consult the local files first. If they encounter a YP escape entry before they find the information for which they are searching, the processes are redirected to the YP service to continue the search.

On a YP slave server, the **/etc/passwd** file should contain only those user entries that are designated for local use only. In addition, an entry for `root` should always be maintained in the local **/etc/passwd** file to permit logging in for maintenance when YP is not running. The final entry should be the YP escape sequence (`+::0:0::`). This entry instructs a system process to consult the YP map at this point. It should be the last entry so that all local entries are checked before the YP map is consulted.

The following is an example of the **/etc/passwd** file that could appear on a YP slave server:

```
root:!:0:0:Sysop:/:/bin/sh
su:!:0:0::/:
daemon:!:1:1::/etc:
bin:!:2:2::/bin:
sys:!:3:3::/usr/sys:
adm:!:4:4::/usr/adm:
adduser:!:0:0::/usr/adm:/etc/adduser
locusr1:!:220:50:Local User 1:/u/locusr1:bin/sh
locusr2:!:221:50:Local User 2:/u/locusr2:/bin/sh
+::0:0::
```

Note: The ! is a special place holder that directs system processes to check an additional security file. This is explained in detail in the section on creating password files.

The **/etc/group** file should contain only those group entries that are designated for local use only. The final entry should be the YP escape sequence (`+:`). This entry instructs a system process to consult the YP map at this point. It should be the last entry so that all local entries are checked before the YP map is consulted.

The following is an example of the **/etc/group** file that could appear on a YP slave server:

```
system:!:0:root,su
printq:!:9:root
locgrp:!:50:locusr1,locusr2
+:
```

The TCP/IP program consults the **/etc/hosts** file before YP is initialized. For this reason, an entry that identifies the local Internet address of the host must appear in the file on a YP slave server. The only other entry required in this file is the **loopback** entry. No other entries are necessary because the YP map is consulted in place of the local **/etc/hosts** file. The entries in **/etc/hosts** should take the following form:

```
Internet_address      hostname
127.0.0.1             localhost
```

Replace the *Internet_address* parameter with the Internet address of the local host, and the *hostname* parameter with the name of the local host. For example, the **/etc/hosts** file on the YP slave server called `slave1` with Internet address `192.200.10.10` would look like the following:

```
192.200.10.10     slave1
127.0.0.1         localhost
```

Transferring Maps From the YP Master Server

In YP, complete replicas of the master YP data base located on the YP master server are transferred to the YP slave servers. The YP slave servers must be authorized to copy files remotely from the YP master server. See the **rcp** (remote copy) command in *Interface Program for use with TCP/IP* for detailed information.

To transfer the replicated YP maps from the YP master server to the YP slave servers, change to the **/etc/yp** directory and issue the following at the command line:

```
cd /etc/yp
ypinit -s master
```

Replace the *master* parameter with the name of the YP master server.

The **ypinit** procedure creates a new subdirectory under **/etc/yp** corresponding to the name of the YP domain you have set. It transfers exact replicas of the maps maintained on the YP master server to this subdirectory. For information on the contents of the maps, see "Creating the YP Maps on the YP Master Server" on page 12-43.

Customizing YP Clients

Customizing YP clients is similar to customizing each YP slave server. Like the YP slave server, customizing work stations to be YP clients involves editing the three local access files that are not replaced by YP maps (**/etc/passwd**, **/etc/group**, and **/etc/hosts**). Follow the procedures listed in "Editing the Local Access Files" on page 12-46.

The final task is to start the **ypbind** process. As discussed previously, the **ypbind** process runs as a background daemon on all YP hosts to find the YP server with a YP map that has the information requested by a client. To start the **ypbind** utility, enter the following at the command line:

```
/etc/ypbind
```

If the entry for **ypbind** is uncommented in the **/etc/rc.nfs** file, **ypbind** starts automatically the next time the system is booted.

Maintaining the YP Environment

It is likely that the YP environment you have configured will require adjustments from time to time. In fact, in large or complex networks, the YP environment may change many times a day. This section discusses how to maintain YP using tools that are supplied with IBM AIX/RT Network File System.

Changing the Default YP Maps

Changing the YP maps to reflect updated system information may be a common occurrence in your daily maintenance tasks. System information, such as a new user account or a changed password, can require constant updating. To modify most YP maps, you will edit the ASCII input file on the YP master server that contains the original YP map, rebuild the YP map, and propagate it to the YP slave servers. The special handling of YP password maps is the exception.

Warning: Except for user password changes made with the **yppasswd** command, YP maps should be modified on the YP master server only. Modifying YP maps on YP slave servers can break the YP service algorithm, which can result in the YP map data being unreliable.

Updating a YP Map

To update the contents of a default YP map, you must first edit the text file that is used as the input file for the map. For example, if you were adding a new machine to your network, you would edit the **/etc/hosts** file by adding an entry that lists the new machine's Internet address and host name.

After editing the input file, change to the **/etc/yp** directory and use the **make** procedure to rebuild the new map by issuing the following at the command line:

```
cd /etc/yp
make map_type
```

The *map_type* parameter specifies the YP maps to be constructed from the input file. For example, issuing the command as `make hosts` causes the host name and the host address YP maps to be rebuilt.

If you have modified several text files or want to confirm that all YP maps are updated, issue the **make** command without parameters. The **make** procedure automatically evaluates every input file on the YP master server. If the file has been modified since the latest YP map for that file was built, the YP map is rebuilt.

The **make** procedure automatically propagates any new maps to the YP slave servers by default. If you prefer not to have the YP maps propagated automatically, add the

NOPUSH parameter to the **make** command sequence. You can assign any non-null value to **NOPUSH**. For example, to have the YP host maps rebuilt but not propagated to the YP slave servers, the following could be entered at the command line:

```
make hosts NOPUSH=noxfr
```

By deferring the transfer process, you can examine the contents of the updated map or test parts of the map before it is transferred.

When you want to transfer the YP maps, you do not have to use the **make** command again. You can transfer the maps manually with the **yppush** command. The **yppush** command transfers a YP map to all hosts listed in a map called **ypservers**. To transfer the maps manually, enter the following at the command line:

/etc/yp/yppush *mapname*

The *mapname* parameter specifies the name of the YP map being transferred.

Updating YP Passwords

Users listed in the YP password map can log in on any host in a particular domain, regardless of whether they are included in the local host's password file. However, unless they are logged in on the YP master server and the YP master server's **/etc/passwd** file is being used for YP map input, users cannot change their password using the AIX **passwd** command. If the **yppasswdd** daemon program is running on the YP master server, users can change their passwords from any host in the domain using the **yppasswd** command.

Note: The **yppasswdd** daemon, which runs on the YP master server only, is usually invoked when the system starts from the **/etc/rc.nfs** file. See "Starting the yppasswdd Daemon" on page 12-44 for more information on running **yppasswdd**.

Activate the **yppasswdd** daemon on the YP master server by entering the following command at the command line:

/usr/etc/rpc.yppasswdd *pw_file* [-m *make_args*]

The *pw_file* parameter specifies the path name of the file being used as the input file for the YP password map. The **yppasswdd** daemon checks the file named by *pw_file* to verify the old password supplied by the user. If verification succeeds (the old password is valid), **yppasswdd** replaces the old password in *pw_file* with the new one.

The flag for the **make** procedure (**-m**) causes the **yppasswdd** daemon to invoke the **make** procedure to rebuild the appropriate YP maps. Include any arguments in place of the *make_args* parameter. Usually, **passwd** is used as an argument to **make**. Variables defined in **/etc/yp/Makefile** can also be assigned as **make** arguments.

In the following example, **/etc/passwd** on the YP master server is used as the input file for the YP password name and ID maps. In this case, the **PASSWD** variable passed to the **make** procedure is set to **/etc/passwd** as follows:

```
/usr/etc/rpc.yppasswdd /etc/passwd -m passwd PASSWD=/etc/passwd
```

When a user enters the **yppasswd** command from any host in the YP domain, the **yppasswdd** daemon changes the **/etc/passwd** file on the YP master server and invokes **make** to rebuild and propagate the password maps.

If the **/etc/yp/passwd** file on the YP master server is used as the input file for the YP password name and ID maps, the **PASSWD** variable is set to **/etc/yp/passwd** as shown in the following:

```
/usr/etc/rpc.yppasswdd /etc/yp/passwd -m passwd PASSWD=/etc/yp/passwd
```

See *AIX Operating System Commands Reference* for more information on the **yppasswd** command.

Adding a New YP Server

If your network configuration grows or changes, it may be necessary to add additional YP slave servers to support the new configuration. Adding a new YP server to your configuration involves modifying the **ypservers** map, which is also used by the system to determine which YP hosts receive the replicas of the YP maps on the YP master server.

The procedure for modifying the **ypservers** map differs from other default maps because no text file is used as input for this map. Instead, the **makedbm** utility is used to create the modified **ypservers** maps.

Modifying the ypservers Map on the YP Master Server

First, list the contents of the current **ypservers** map using the **makedbm** command with the **−u** flag. The output of this command, along with the echoed name of the new server, is then used as input to another **makedbm** command that creates the new map temporarily called `tmpsvrs`. Change to the **/etc/yp** directory and enter the following:

```
cd /etc/yp
(makedbm -u domain/ypservers ; echo new_server) | makedbm - tmpsrvrs
```

The *domain* parameter specifies the name of the YP domain (and the directory where the YP maps are kept). The *new_server* parameter specifies the name of the host being added to the **ypservers** map.

You can check the contents of the new map by entering the following at the command line:

```
makedbm -u tmpsrvrs
```

When you have verified that the new server is included in the **tmpsrvrs** map, use the **mv** (move) command to replace the old **ypservers** map files with the new ones. Enter the following at the command line:

```
mv tmpsrvrs.pag domain/ypservers.pag
mv tmpsrvrs.dir domain/ypservers.dir
```

Modifying the ypservers Map on the YP Slave Server

After you have added the new server to the **ypservers** map on the YP master server and updated it, run the **ypinit** procedure on the new server to have all of the YP maps replicated on it. Change to the **/etc/yp** directory and enter the following at the command line:

```
cd /etc/yp
ypinit -s master
```

The *master* parameter specifies the name of the YP master server.

Note: You must also start the **ypserv** process on the new YP slave server. See "Customizing YP Slave Servers" on page 12-45 for information.

Creating a New YP Map

User information requirements at your site may make it necessary to add new maps to service your YP domain. Standard text files can be created or filtered through processes such as **awk**, **grep**, and **sed**, and passed as input to the **makedbm** utility that creates the YP maps. Consult the existing **/etc/yp/Makefile** for examples. It is recommended that you use mechanisms similar to those in the **Makefile** when creating the new maps.

After the new map is created on the YP master server, it must be transferred to the YP slave servers to maintain data base consistency throughout the network in one of the following ways:

- With the **yppush** command. This is issued from the YP master server to push the new map from the YP master server to the YP slave servers.
- With the **ypxfr** command. This issued from a YP slave server to get the new map from the YP master server.

To provide ongoing maintenance for the new map (such as updating), create an entry for the new map in **/etc/yp/Makefile**.

The following example shows how to create a new map called `udir.nam` that lists the home directories of users in a YP domain. The new map is created using filtered input from the YP password file (in this case, **/etc/yp/passwd**). The keys for the map are the user names, and their corresponding values are the users' home directories. To begin creating the map, change to the **/etc/yp** directory and enter the following at the command line:

```
cd /etc/yp
awk '{FS=":" ; OFS="\t" ; print $1,$6}' /etc/yp/passwd |
     \ makedbm - domain/udir.nam
```

The *domain* parameter specifies the current YP domain.

After the new map `udir.nam` is propagated to the YP slave servers, any YP client in the YP domain can query it for information with the **ypcat** or **ypmatch** commands.

Keeping Maps on YP Slave Servers Current

To ensure the information in YP maps is reliable and consistent throughout a YP domain, all updates to the maps located on the YP master server must be propagated to the YP slave servers. When a map on the YP master server is updated, the **make** procedure automatically executes the **yppush** utility after rebuilding the map. The **yppush** command notifies all YP slave servers that a map must be transferred, and the **ypserv** process on each YP slave server invokes the **ypxfr** command to get the updated map. However, a YP slave server that is out of service when **yppush** sends notice of the new map does not invoke the **ypxfr** command, which means it retains the earlier version of the YP map when it returns to the network. To prevent such situations, each YP slave server can be set to request updated maps from the YP master server at regular intervals using the **cron** process.

Some maps require update verifications more frequently than others. For example, the password and mail alias maps can change many times a day, and must be checked more frequently than the protocols or services maps, which may not change for months at a time.

The **ypxfr** command requests a single map. To avoid having **crontab** entries for each map, you can group **ypxfr** commands for several maps in a shell script, and have the **cron** process execute the shell script at appropriate intervals. Group the maps together according to how often they need updating. The following shell scripts exist in the **/etc/yp** directory:

- **ypxfr_1perh** for checking each hour
- **ypxfr_1perd** for checking once a day
- **ypxfr_2perd** for checking twice a day.

Modify these shell scripts as necessary to meet the requirements of your site.

Use the **crontab** command to have the shell scripts executed automatically at specific intervals.

Note: Before you make new entries for the **ypxfr** scripts, it is important to copy the current **crontab** entries for **root** to a temporary file. This prevents the old entries from being lost when **cron** processes the new file. To create a temporary file to hold the current entries, enter the following at the command line:

`crontab -l >` *filename*

The *filename* parameter specifies the temporary file you have created.

Using a text editor, add your new entries for the **ypxfr** procedures to the temporary file. Modify the execution schedule for each entry according to how often each **ypxfr** script should be run, as in the following examples:

- To execute the **ypxfr_1perh** script at 37 minutes after every hour of the day:

  ```
  37 * * * * /etc/yp/ypxfr_1perh
  ```

- To execute the **ypxfr_2perd** script at 11:00 a.m. and 11:00 p.m. each day:

  ```
  0 11,23 * * * /etc/yp/ypxfr_2perd
  ```

- To execute the **ypxfr_1perd** script at 3:30 a.m. each day:

  ```
  30 3 * * * /etc/yp/ypxfr_1perd
  ```

Note: It is recommended that the request times differ for each YP slave server to prevent overloading the YP master server with requests from all YP slave servers at the same time.

After the entries have been made, issue the following at the command line to have **cron** read the file and execute the procedures at the appropriate times:

`crontab` *filename*

Although the **ypxfr** command can also be issued at the command line, this use should be reserved for special circumstances such as when creating a test YP environment or bringing a YP slave server into service after it has been off-line.

You can maintain a record of the **ypxfr** activity in a log file you can create and call `/etc/yp/ypxfr.log`. (Remember to trim this file periodically in the same manner as you trim your other administrative logs.) To discontinue **ypxfr** logging, remove the `/etc/yp/ypxfr.log` file.

Yellow Pages Entries in Non-YP Files

Two files used to validate remote access from other hosts, **/etc/hosts.equiv** and **/.rhosts**, are not used as input for YP maps, but these files can have entries that refer to the Yellow Pages. To direct system processes to consult YP from these files, use one of the following:

- A single + on a line by itself, which directs a system process to consult YP for all accesses

- A `+@` or `-@` appearing before the name of a network group, which permits or restricts access by members of that network group.

Using a single + as an entry in the **hosts.equiv** file means that only the **/etc/passwd** file (and associated YP map, if being used) is consulted to validate users remotely logging in on this system. Users listed in the password file or YP map are permitted to log in without a password.

Note: It is not recommended to have unrestricted access in the **/.rhosts** file (**.rhosts** for the root user). The single + entry effectively permits any root user from another host to log in as root on this system without a password.

The following example demonstrates the use of netgroup entries in the **hosts.equiv** and **/.rhosts** files:

```
+@netgroup1
-@netgroup2
```

The hosts listed in **netgroup1** are considered trusted. This means users attempting to log in remotely from a host in **netgroup1** can log in without a password if they are listed in **/etc/passwd** or the associated YP map. The hosts listed in **netgroup2**, on the other hand, are not trusted. This means that users attempting to log in remotely from a host in **netgroup2** must supply their password.

See *AIX Operating System Technical Reference* for information on the **netgroup** file format.

Interpolation of Entries Between Local Files and YP Maps

You can direct system processes that consult the local files **/etc/passwd** and **/etc/group** to use certain information from the local file while using other information from the associated YP map. The plus (+) symbol is placed at the beginning of a file entry to direct the system process to use the values from the fields in the associated YP map for those fields left blank in the entry while using the local values that are specified.

For example, an entry in the **/etc/passwd** file usually appears in the following form:

user_namepassword:userid:groupid:*account_information*:*home_directory*:*login_shell*

A system process reads the values of these fields locally. To have the system process interpolate entries from the YP map and the local file, the entry could appear as follows:

+*username*:::*account_information*:*home_directory*:

In this example, the entry directs the system process to get the password, user ID, group ID, and login shell values from the fields in the associated YP password map, but to use the *additional_data* and *home directory* values listed in the local file. Another way to think of it is that the local values for the *additional_data* and *home_directory* override the values in the YP map.

Note: The user ID (UID) and the group ID (GID) numerical entries cannot be overridden locally if YP interpolation is used on a password file or group file entry.

In this example, the following entry in the **/etc/group** file directs the system process to get the group password and GID fields for the review group from the YP, but defines `joe` and `jane` as the members of the group, which overrides any members listed in the corresponding entry in the YP map:

```
+review:::joe,jane
```

Network group entries can also be included by using the +@ with the network group name. In the following example entry, all users in the novices netgroup use the locally defined **GCOS** and login shell fields while other fields are obtained from the YP for each user in the netgroup:

```
+@novices::::A Novice User::/usr/local/help_menu
```

The +@ and -@ signifiers can also be used in combination with netgroup names in the **/etc/hosts.equiv** and **/.rhosts** files as discussed in "Yellow Pages Entries in Non-YP Files" on page 12-54.

Appendix A. Printer Control Codes

Figure A-1 on page A-2 lists the printer control codes that you can use with RT printers. If the printer can perform the function by itself, the system passes the code directly to the printer. If the codes do not work on the printer installed on your system, the printer support system performs one of the following actions:

- Tries to emulate the control with functions that the installed printer does have.
- Removes the control from the output stream. The function is not performed.

The three code columns in the table show different representations of the same code, depending on how you enter the code into the data stream.

Control Name This column shows the name of the control character. In many cases the name is the same as the keyboard key that produces the required ASCII code for the control code.

Hex Code This column show the hexadecimal representation of the control code.

ASCII Code This column shows the decimal representation of the control code.

Category	Function Performed	Control Name	Hex Code	ASCII Code
Control:	Provides a null value. Used as a list terminator.	NUL	00	0
	Sounds the buzzer. [2]	BEL	07	7
	Prints the next character as a printable character. The next character is a control with an ASCII value of less than 32.	ESC ^	1B5E	27 94
	Prints more than one character with an ASCII value that is below 32.	ESC \ *m n c*	1B5C *m n c*	27 92 *m n c*
	Clears the printer memory of all data waiting to be printed following the last received line feed. [1]	CAN	18	24
	Sets resolution for raster image print (*n* indicates a string of control bytes).[1]	ESC [O *n*	1B5B4F *n*	27 91 79 *n*
	Performs a printer power on reset.[1]	ESC [K	1B5B4B 0100 0	27 91 75 1 0 0
Positioning the Printhead:	Sets back space.	BS	08	8
	Sets horizontal tab.	HT	09	9
	Sets horizontal tabs (*n* is a list of one or more tab positions).	ESC D *n* NUL	1B44*n*00	27 68 *n* 0

Figure A-1 (Part 1 of 7). Printer Control Codes

1 Use with the Color Jet Printer.

2 Do not use these controls when using the print queue.

3 These controls may not work on the installed printer. Use *passthrough* mode to send these codes to the printer.

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Sets tab stops to power-on settings.	ESC R	1B52	27 82
	Sets line feed.	LF	0A	10
	Sets reverse line feed.	ESC]	1B5D	27 93
	Starts automatic line feed.	ESC 5 1	1B35 1	27 53 1
	Stops automatic line feed.	ESC 5 0	1B35 0	27 53 0
	Provides a carriage return (no line feed).	CR	0D	13
	Provides a vertical tab.	VT	0B	11
	Sets vertical tabs (*n* is a list of tab positions).	ESC B *n* NUL	1B42 *n* NUL	27 66 *n* NUL
Paper Control:	Provides a form feed.	FF	0C	12
	Sets top of forms. [2]	ESC 4	1B34	27 52
	Ignores End of Forms. [2]	ESC 8	1B38	27 56
	Respects End of Forms. [2]	ESC 9	1B39	27 57
	Sets skip perforation [2] (*n* is lines to skip).	ESC N *n*	1B4E *n*	27 78 *n*
	Stops skip perforation [2]	ESC O	1B4F	27 79
Formatting the Page Image:	Uses 12 characters-per-inch printing.	ESC :	1B3A	27 58
	Sets 1/8″ Line spacing.	ESC 0	1B30	27 48
	Starts *n*/72″ Line spacing.	ESC 2	1B32	27 50
	Sets *n*/72″ Line spacing.	ESC A *n*	1B41 *n*	27 59 *n*

Figure A-1 (Part 2 of 7). Printer Control Codes

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Sets Page Length. [2] (*n* is lines per page).	ESC C *n*	1B43 *n*	27 67 *n*
	Sets Page Length [2] (*n* is inches per page).	ESC C 0 *n*	1B43 0 *n*	27 67 0 *n*
	Sets left and right margins (*m* and *n* are column numbers).	ESC X *m n*	1B58 *m n*	27 88 *m n*
	Sets top and bottom margins (*m* and *n* are length of control; *t1 t0* are high/low-order bytes of top margin; *b1 b0* are high/low-order bytes of bottom margin).	ESC [S *m n t1 t0 b1 b0*	1B5B53 *m n t1 t0 b1 b0*	27 91 83 *m n t1 t0 b1 b0*
	Starts automatic line justification.	ESC M 1	1B4D 1	27 77 1
	Stops automatic line justification.	ESC M 0	1B4D 0	27 77 0
	Starts proportional spacing.	ESC P 1	1B50 1	27 80 1
	Stops proportional spacing.	ESC P 0	1B50 0	27 80 0
	Sets print resolution for draft quality font and ESC K image data (*n* indicates the resolution value).[1]	ESC [0 *n*	1B5B30 0100 *n*	27 91 48 1 0 *n*
	Sets presentation surface color (*n* indicates the available colors).[1]	ESC [L *n*	1B5B4C 0200 00 *n*	27 91 76 2 0 *n*
Controlling the Ribbon:	Sets color band 1 (yellow).	ESC y	1B79	27 121
	Sets color band 2 (magenta).	ESC m	1B6D	27 109
	Sets color band 3 (cyan).	ESC c	1B63	27 99

Figure A-1 (Part 3 of 7). Printer Control Codes

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Sets color band 4 (black).	ESC b	1B62	27 98
	Sets automatic ribbon band shift.	ESC a	1B61	27 97
	Sets the background color for Text, ESC K, or ESC [0 printing (*n* indicates the available colors).[1]	ESC [N *n*	1B5B4E 0100 *n*	27 91 78 1 0 *n*
	Sets the foreground color for Text, ESC K, or ESC [0 printing (*n* indicates the available colors).[1]	ESC [M *n*	1B5B4D 0100 *n*	27 91 77 1 0 *n*
	Swaps the foreground and background printing color ($n=0$ indicates normal printing; $n=1$ indicates reversing background and foreground colors).[1]	ESC []	1B5B5D 0100 *n*	27 91 93 1 0 *n*
Selecting Print Mode:	Starts double wide printing.	SO	0E	14
	Stops double wide printing.	DC4	14	20
	Starts double wide continuous printing.	ESC W 1	1B57 1	27 87 1
	Stops double wide continuous printing.	ESC W 0	1B57 0	27 87 0
	Starts compressed printing.	SI	0F	15
	Stops compressed printing.	DC2	12	18
	Starts underline printing.	ESC -1	1B2D 1	27 45 1
	Stops underline printing.	ESC -0	1B2D 0	27 45 0
	Starts emphasized printing.	ESC E	1B45	27 69
	Stops emphasized printing.	ESC F	1B46	27 70

Figure A-1 (Part 4 of 7). Printer Control Codes

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Starts double strike printing.	ESC G	1B47	27 71
	Stops double strike printing.	ESC H	1B48	27 72
	Starts superscript printing.	ESC S 0	1B53 0	27 83 0
	Starts subscript printing.	ESC S 1	1B53 1	27 83 1
	Stops superscript or subscript printing.	ESC T	1B54	27 84
	Starts color underline, bypassing white space (*n* defines the available color).[1]	ESC [B 1 *n*	1B5B42 0200 1 *n*	27 91 66 2 0 1 *n*
	Starts continuous color underline (*n* defines the available color).[1]	ESC [B 0 *n*	1B5B42 0200 0 *n*	27 91 66 2 0 0 *n*
	Stops underline.	ESC [E	1B5B450000	27 91 69 0 0
Selecting the Character Set:	Uses PC character set 2.	ESC 6	1B36	27 54
	Uses PC character set 1.	ESC 7	1B37	27 55
	Selects font (*n* specifies the font; varies with printer type).	ESC I *n*	1B49 *n*	27 73 *n*
	Sets graphic set ID (*c* selects graphic set 0, 1 or 2).	ESC [T 1 0 *c*	1B5B54 1 0 *c*	27 91 84 1 0 *c*
Using Bit Image Graphics: [3]	Sets bit graphics normal (*n* is a string of control bytes).	ESC K *n*	1B4B *n*	27 75 *n*
	Sets graphics dual-half speed (*n* is a string of control bytes).	ESC L *n*	1B4C *n*	27 76 *n*

Figure A-1 (Part 5 of 7). Printer Control Codes

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Sets bit graphics dual-normal speed (*n* is a string of control bytes).	ESC Y *n*	1B59 *n*	27 89 *n*
	Sets bit graphics high-half speed (*n* is a string of control bytes).	ESC Z *n*	1B5A *n*	27 90 *n*
	Sets aspect ratio to 1:1.	ESC n 1	1B6E 1	27 110 1
	Sets aspect ratio to 5:6.	ESC n 0	1B6E 0	27 110 0
	Moves carriage to home position.	ESC <	1B3C	27 60
	Moves right *n*/120.	ESC d *n*	1B64 *n*	27 100 *n*
	Moves left *n*/120.	ESC e *n*	1B65 *n*	27 101 *n*
	Starts unidirectional printing.	ESC U 1	1B551	27 85 1
	Stops unidirectional printing.	ESC U 0	1B550	27 85 0
	Sets 7 dot line spacing.	ESC 1	1B31	27 49
	Sets graphics line spacing (*n* is the number of 1/216-inch steps).	ESC 3 *n*	1B33 *n*	27 51 n
	Sets variable space line feed (*n* is the number of 1/216-inch steps).	ESC J *n*	1B4A *n*	27 74 *n*
	Moves vertical presentation down the page in number of dots (*n* indicates how far to move the presentation). [1]	ESC [U *n*	1B5B55 0100 *n*	27 91 85 1 0 *n*
Selecting a Printer:	Selects the printer to accept data. [2]	DC1	11	17
	Deselects the printer so as to not receive data. [2]	DC3	13	19

Figure A-1 (Part 6 of 7). Printer Control Codes

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Sets initialize function on. [2]	ESC ? 1	1B3F 1	27 63 1
	Sets initialize function off. [2]	ESC ? 0	1B3F 0	27 63 0
	Queries a parallel attached printer for identification. If the device queried is equal to *n*, this printer deactivates the select line. [2]	ESC Q *n*	1B51 *n*	27 81 *n*

Figure A-1 (Part 7 of 7). Printer Control Codes

1 Use with the Color Jet Printer.

2 Do not use these controls when using the print queue.

3 These controls may not work on the installed printer. Use *passthrough* mode to send these codes to the printer.

Appendix B. Distributed Services Customization Forms

This appendix provides forms to help define your network. Copy these forms as required for each node in the network, and then enter the information on the forms before defining the network. See "Configuring a Basic Distributed Services System" on page 11-17 and "Using Distributed Services Menus" on page 11-93 for explanations of the information needed on the forms.

DEFINING A STATIC NODE

NODE: ________

Remote Nickname:	☐	None (default)	☐	____________
Remote Node ID:	____________			
Node Security:	☐	None (default)		
	☐	Secure		
		Password:[1]		____________
Data Link Type:	☐	Ethernet (default)	☐	SDLC
	☐	Token Ring	☐	802.3

1 Do not write the password in this space if unauthorized people have access to this form. Password is either 16 hexadecimal characters or 30 to 80 alphameric characters.

DEFINING A DYNAMIC NODE

NODE: ________

Remote Nickname:	____________
Remote Node ID:	____________
Node Security:	☐ None (default)
	☐ Secure
	Password:[2] ____________
Prototype Connection Profile:	____________
Prototype Physical Link Profile:	____________
Command/Selection Sequence:	____________

2 Do not write the password in this space if unauthorized people have access to this form. Password is either 16 hexadecimal characters or 30 to 80 alphameric characters.

INBOUND NETWORK ID TRANSLATIONS

NODE: ________

User IDs:

Network ID	Originating Node (* = Any Node)	Local ID

INBOUND NETWORK ID TRANSLATIONS

NODE: ________

Group IDs:

Network ID	Originating Node (* = Any Node)	Local ID

OUTBOUND NETWORK ID TRANSLATIONS

NODE: _________

User IDs:

User Name	Local User ID	Network ID

OUTBOUND NETWORK ID TRANSLATIONS

NODE: --------

Group IDs:

User Name	Local User ID	Network ID

FILE SYSTEM MOUNTS

NODE: ________

Mount Point (Local Path Name)	Location (nodename =)	Remote Path Name (dev =)	Auto Mount Type (type =)

Appendix C. Token-Ring Diagnostics

The Token-Ring diagnostics provides error analysis, error reporting, and status information on the Token-Ring network. Reporting is done for both hard and soft errors.

Hard errors are permanent faults, usually in equipment, that cause the Token-Ring to stop operating within the normal Token-Ring network architecture protocols. A Token-Ring station downstream from the hard fault recognizes a hard error at the receiver side of its attachment. You can bypass this error and restore the Token-Ring to an operational state by reconfiguring the Token-Ring. To restore the Token-Ring to full operation, repairs may be required.

When a Token-Ring station detects a hard error failure, it transmits frames reporting this condition, called ***beaconing***, until its input signal is restored, or until it removes itself from the Token-Ring. You are notified of this condition with error message `010-200 E Ring not working` as described in the "Messages from the Data Messages Area" on page C-6.

Soft errors are intermittent faults that temporarily disrupt operation of the Token-Ring network and are normally tolerated by error recovery procedures. Soft errors are indicated by architectural inconsistencies in received or repeated frames and by a Token-Ring station's inability to process received frames. Each Token-Ring station maintains a set of counters to measure the frequency of occurrence of the most critical soft errors.

Isolating errors are types of soft errors that can isolate the most probable source of the soft error to the reporting station, its nearest active upstream neighbor, and the media between them. For these soft errors, an analysis is done to determine if these errors are occurring at a rate that significantly degrades Token-Ring performance. When such a condition is detected, the message `010-110 E Ring error limit exceeded` displays.

Non-isolating errors are another type of soft errors that only isolate an error to the Token-Ring on which they occur. If one of these soft error counters exceeds a threshold value, the message `010-104 I  Recovered error counter` displays, and the error counter resets. No other analysis is performed on these soft errors.

For more detailed information about hard and soft errors on the Token-Ring network and how they are detected and analyzed, see the "Ring Error Monitor" chapter in the *Token-Ring Network Architecture Reference*. More detailed information about variable data presented with the data messages can also be found in this manual.

The Token-Ring diagnostics requires that the Block I/O AIX Device Driver, the Token-Ring VRM Device Driver, and the IBM PC Network Adapter be installed. After installing the Token-Ring diagnostics, run the **devices** command and add the Token-Ring adapters. To allow reporting of beaconing conditions change the value of the Token-Ring adapter **opnop** (open options) field to 3580 with the **devices** command.

To configure the block I/O kernel device driver to run with the desired Token-Ring adapters, edit the **/etc/biodd** file to add the device names of Token-Ring adapter cards and then run the **biodd_cfg** command. If you edit the **/etc/rc.include** file and uncomment the line

```
# /etc/biodd_cfg
```

biodd_cfg runs each time you start the system.

If you are running in an active service mode in a code-service environment on a client machine and wish to run the Token-Ring diagnostics, copy the file **/usr/lpp/trd/samples/trdconf** on the server to the file **/etc/trdconf** on the client.

Before you start the Token-Ring diagnostics, open a virtual terminal. This lets you monitor the Token-Ring network without dedicating the system to the Token-Ring diagnostics. To start the Token-Ring diagnostics, enter:

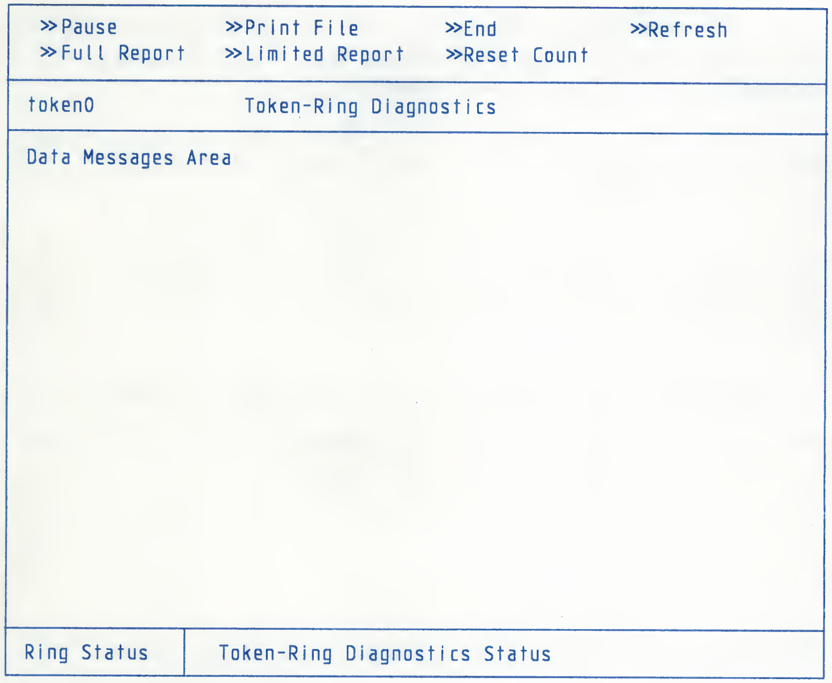
```
trdiag
```

For information on the **trdiag** command, see *AIX Operating System Commands Reference*. You can run the Token-Ring diagnostics on more than one adapter by opening a separate virtual terminal for each adapter. The **trdiag** command starts the diagnostics, and displays the Token-Ring diagnostics copyright screen. Press the Usability Services keyboard **Quit** or **Select** keys to display the Token-Ring Diagnostics screen, Figure C-1 on page C-3.

You provide input to the Token-Ring diagnostics with the commands shown on the Token-Ring Diagnostics screen. To select a command, move the cursor to the command name and press the **Select** function key. If you add:

```
ALTNUM=yes
export ALTNUM
```

to your **.profile**, or to **/etc/profile** numbers display with the commands. With the number displayed you can select a command by pressing the **Alt** key while you press the key number shown with the command. The numbers cannot be from the numeric key pad. When you select a command, the command name displays in reverse video indicating the function is enabled. When you select the same command again, the function is disabled, and the reverse video is removed. An `operation in progress` message displays briefly in the Token-Ring Diagnostics Status area of the screen while the command is being performed.



A5ACG026

Figure C-1. Token-Ring Diagnostics Screen

The commands function as follows:

Pause

Starts and stops the scrolling of messages in the Data Messages area. When the Data Messages area fills, it starts scrolling. Select the **Pause** command to stop scrolling. Selecting the **Pause** command a second time restarts scrolling.

Print File

Starts or stops sending messages from the Data Messages area to a temporary print file. The initial state of the **Print File** command is not to send messages to the print file. When you select the **Print File** command, all messages currently displayed in the Data Messages area and all subsequent messages are written to the print file. Selecting **Print File** a second time sends the print file to the printer queue and then deletes the print file.

Full Report

Starts and stops the display of all soft error report messages in the Data Messages area. In addition, displays messages for neighbor notification failure, ring monitor error, and recovered error counter (for nonisolating type errors). The initial state of the **Full Report** command is not to send these messages.

Limited Report
Starts and stops the display of soft error report messages from those adapters on which the number of errors exceeds the halfway point to the Token-Ring error limit. The initial state of this command is not to display those soft error report messages.

Refresh
Updates the screen with current information from the Token-Ring diagnostics. This command performs its function to completion.

Reset Counts
Resets the Token-Ring diagnostics error counters without losing any messages that have not been displayed. This command performs its function to completion. This is equivalent to restarting the Token-Ring diagnostics.

End
Stops the Token-Ring diagnostics. This command performs its function to completion.

To get help information about any command name, move the cursor to the command and press the **Help** function key.

Output

Output from the Token-Ring diagnostics displays in three different areas of the screen. The three areas are:

- Data Messages area
- Ring Status area
- Token-Ring Diagnostics Status area.

The output to the Data Messages area is always logged to a disk file and optionally logged to a temporary print file by selection of the **Print File** command. The disk file name is specified with the **-t** flag or from the `file =` parameter of the `trdlog` stanza of the **/etc/trdconf** file. For more information about the **/etc/trdconf** file, see **trdconf** in *AIX Operating System Technical Reference*.

Data Messages Area

The Data Messages area displays the most detailed information about the Token-Ring. Messages in this area include warnings and error messages sent as a result of the error analysis and informational messages indicating changes in Token-Ring parameters or the Token-Ring status. All messages in this area contain the following:

- Message date and time in the format:

  mm/dd/yy hh:mm:ss:

where

  - *mm* is the month number
  - *dd* is the number of the day of the month
  - *yy* is the last two numbers of the year
  - *hh* is the hour in the day (using the 24-hour clock)
  - *mm* is the minute number
  - *ss* is number of seconds.

- Message number in the format:

  010-*aaa t*

where

  - **010** is the three-digit code assigned to Token-Ring diagnostics
  - *aaa* is the three-digit message number
  - *t* one of the following message types:
    - **I** for information messages
    - **W** for warning messages
    - **E** for error messages.

- Message text containing a one-line description of the problem.
- Variable data about the message.

  Variable data is not shown for all messages. If a message has variable data, it starts on the line following the message. Variable data is generally displayed in hexadecimal with the data fields separated by blanks.

Messages from the Data Messages Area

The following messages are written to the log file and to the Data Messages area unless otherwise noted. For information on termination error messages, see *IBM RT Messages Reference.*

010-001 I Ring diagnostics started

Cause: The Token-Ring diagnostics successfully opened the Token-Ring adapter and completed its initialization. `Normal` displays in the Ring Status area for this message.

Action: This message is for information only.

Example Data:

```
01/13/88  08:26:29 010-001 I  Ring diagnostics started
--------->  7FFF8100088A 02.05.0000

            This          Token-Ring
            Adapter's     Diagnostics
            Address       Version/Release/Level
```

A5ACG027

010-010 I Full error reporting enabled

Cause: You selected the **Full Report** command or specified the **-f** flag with **trdiag**.

Action: This message is for information only.

Example Data:

```
01/12/88  06:58:10 010-010 I Full error reporting enabled
```

010-011 I Full error reporting disabled

Cause: You selected the **Full Report** command while full error reporting was enabled.

Action: This message is for information only.

Example Data:

```
01/12/88  06:59:10 010-011 I Full error reporting disabled
```

010-012 I **Limited error reporting enabled**

Cause: You selected the **Limited Report** command or specified the **-l** flag with **trdiag**.

Action: This message is for information only.

Example Data:

```
01/13/88  05:12:06 010-012 I Limited error reporting enabled
```

010-013 I **Limited error reporting disabled**

Cause: You selected the **Limited Report** command while limited error reporting was enabled.

Action: This message is for information only.

Example Data:

```
01/13/88  05:59:52 010-013 I Limited error reporting disabled
```

010-014 I **Error counters reset**

Cause: You selected the **Reset Counts** command resetting all error counters to 0. A Ring Status of `Soft Error` changes to `Normal` when this message displays.

Action: This message is for information only.

Example Data:

```
01/13/88  05:59:54 010-014 I Error counters reset
```

010-040 E **Ring diagnostics initialization failed**

Cause: The Token-Ring Diagnostics did not start successfully.

Action: This message is for information only. It is not displayed on the screen but is written to the log file. It is followed in the log file by message `010-306`.

Example Data:

```
01/13/88  07:01:29 010-040 E Ring diagnostics initialization failed
```

010-049 I **Ring diagnostics ended**

Cause: You selected the **End** command to stop the Token-Ring diagnostics or an error occurred that caused the program to stop.

Action: This message is for information only. It is not displayed on the screen but is written to the log file. If the Token-Ring diagnostics ends because of an error, this message is preceded by message 010-306 which contains information about the cause of the error.

Example Data:

```
01/13/88  07:02:06 010-049 I Ring diagnostics ended
```

010-085 W **Receiver congestion**

Cause: The reporting adapter cannot receive a significant number of frames from the Token-Ring because the input buffer is full. The problem may not be in the adapter receiving the data, but may be in one or more adapters sending an excessive number of frames to the congested adapter.

Action: This message is for information only. When the rate of received frames decreases, normal processing continues.

Example Data:

```
01/13/88  06:03:46 010-085 W  Receiver congestion
--------->  7FFF81000093
            ____________
                 |
              Reporting
              Adapter's
              Address
```

A5ACG028

010-086 I **Receiver congestion ended**

Cause: The rate of received frames decreased and normal processing continues.

Action: This message is for information only.

Example Data:

```
01/13/88  06:09:25 010-086 I  Receiver congestion ended
--------->  7FFF81000093
            ____________
                 |
              Reporting
              Adapter's
              Address
```

A5ACG029

010-101 E **Ring error limit exceeded**

Cause: The number of soft errors exceeds the allowable limit. This could cause a noticeable degradation in the performance of the Token-Ring network. `Soft Error` displays in the Ring Status area with this message.

Action: See the *Token-Ring Network Problem Determination Guide.*

Example Data:

```
01/13/88  08:33:02 010-101 E  Ring error limit exceeded
--------->  522D2D203137 00000000 80  522D2D203138 00000000 80
            ____________ ________ __  ____________ ________ __
                /          /       |        |          |      \
      First       First       Error  Second      Second      Error
      Adapter's   Adapter's   Count  Adapter's   Adapter's   Count
      Address     Physical           Address     Physical
                  Address                        Address
```

A5ACG030

Note: If the variable data for the first adapter are not available, they are replaced with two asterisks. This indicates that the Token-Ring diagnostics isolated the problem to the adapter address shown.

010-102 W Ring errors increasing

Cause: The adapters on the Token-Ring are recording soft errors, and the number is approaching an unacceptable level. `Normal` displays in the Ring Status area for the message.

Action: This message is for information only.

Example Data:

```
01/13/88  08:33:00 010-102 W  Ring errors increasing
--------->  522D2D203137 00000000 5A  522D2D203138 00000000 5A

First       First       Error  Second      Second      Error
Adapter's   Adapter's   Count  Adapter's   Adapter's   Count
Address     Physical           Address     Physical
            Address                        Address
```

A5ACG031

Note: If the variable data for the first adapter are not available, they are replaced with two asterisks. This indicates that the Token-Ring diagnostics isolated the problem to the adapter address shown.

010-103 I Ring errors decreasing

Cause: The rate of soft errors is decreasing. The source of the errors may have been removed. `Normal` displays in the Ring Status area for this message.

Action: This message is for information only.

Example Data:

```
01/13/88  08:33:00 010-103 I  Ring errors decreasing
--------->  522D2D203137 00000000 7E E522D2D203138 00000000 7E

First       First       Error  Second      Second      Error
Adapter's   Adapter's   Count  Adapter's   Adapter's   Count
Address     Physical           Address     Physical
            Address                        Address
```

A5ACG032

Note: If the variable data for the first adapter are not available, they are replaced with two asterisks. This indicates that the Token-Ring diagnostics isolated the problem to the adapter address shown.

010-104 I **Recovered error counter**

Cause: The number of errors, for a type of error, exceeded the reporting limit and the error counter is reset. This message displays only when full error reporting is enabled and can be expected as a normal function of the Token-Ring. `Normal` or `Soft Error` displays in the Ring Status area for this message.

Action: This message is for information only. No analysis is done on this error as it cannot be isolated to one adapter.

Example Data:

```
01/13/88  08:38:34 010-104 I  Recovered error counter
--------->  05  46
           /     \
      Error     Error
      Type      Count
```

A5ACG033

where Error Type is:

01	Lost frame count
02	Receiver congestion error count
03	Frame copied error count
04	Frequency error count
05	Token error count
06	Reserved
07	Table full error count
08	Minimum decrement error count

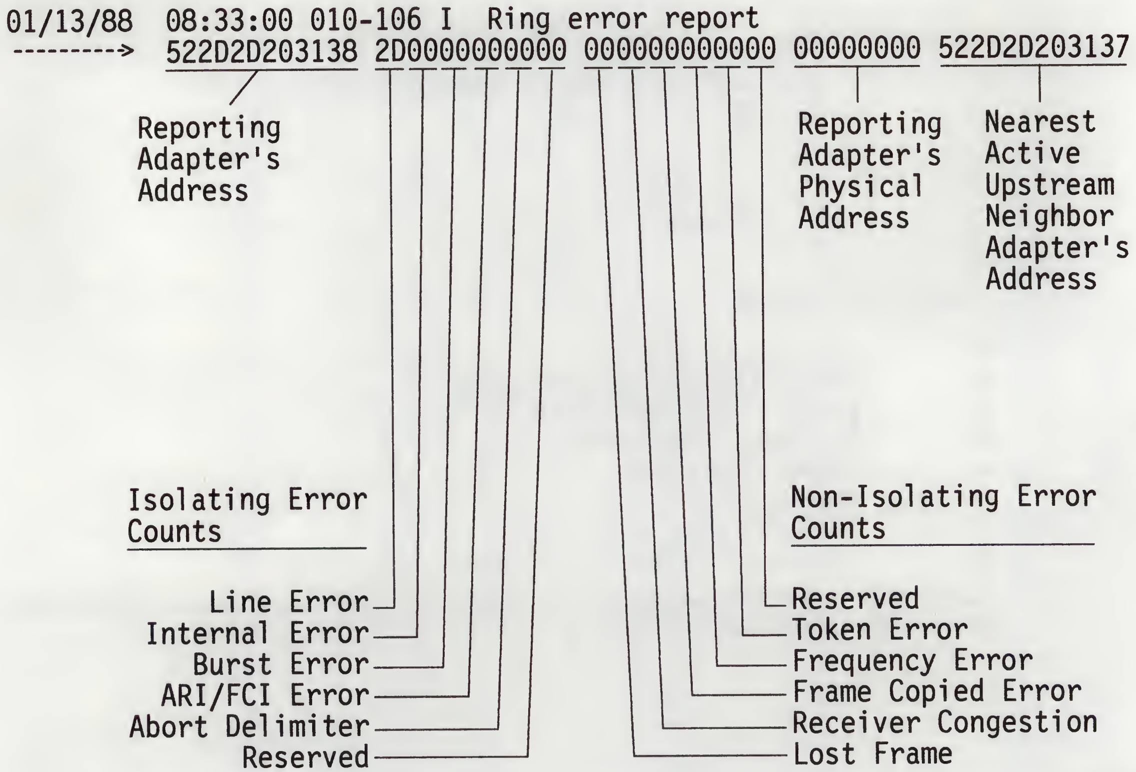
Note: For more information about variable data, see the *Token-Ring Network Architecture Reference.*

010-106 I Ring error report

Cause: A soft error is detected. This message only displays when the **Full Report** or **Limited Report** command is enabled. `Normal` or `Soft Error` displays in the Ring Status area for this message.

Action: This message is for information only.

Example Data:



A5ACG034

Note: For more information about variable data, see the *Token-Ring Network Architecture Reference.*

010-108 I Neighbor notification failure

Cause: The Token-Ring neighbor notification process encountered an error, and recovery has taken place. This message displays only when the **Full Report** command is enabled. `Normal` or `Soft Error` displays in the Ring Status area for this message.

Action: This message is for information only.

Example Data:

```
01/13/88  08:31:46 010-108 I  Neighbor notification failure
--------> 10005A00573E 10005A001391
           /                |
          Ring           Address of
          Monitor's      Last Adapter
          Address        Responding to
                         Neighbor Notification
```

A5ACG035

010-109 I Ring monitor error - ring recovered

Cause: The Token-Ring recovered after a ring monitor error occurred. This message displays only when the **Full Report** command is enabled. `Normal` or `Soft Error` displays in the Ring Status area for this message.

Action: This message is for information only.

Example Data:

```
01/13/88  08:31:46 010-109 I  Ring monitor error - ring recovered
--------> 10005A00573E 0001 00000000 10005A001391
           /            |       \          \
          Ring        Error   Ring        Nearest Active
          Monitor's   Reason  Monitor's   Upstream Neighbor
          Address     Code    Physical    Adapter's
                              Address     Address
```

A5ACG036

Note: For more information about the meaning of the variable data, see "monitor error" in the index of the *Token-Ring Network Architecture Reference.*

010-140 I **Only adapter on ring**

Cause: The Token-Ring diagnostics detected that this is the only active adapter on the Token-Ring.

Action: If other devices are known to be active on the Token-Ring, see Part 1 *Token-Ring Network Problem Determination Guide.*

Example Data:

```
02/05/88 08:31:46 010-140 I  Only adapter on ring
```

010-141 I **Additional adapter(s) on ring**

Cause: The Token-Ring diagnostics detected that at least one other adapter on the Token-Ring has become active.

Example Data:

```
02/05/88 08:33:46 010-141 I  Additional adapter(s) on ring
```

Action: This message is for information only.

010-200 E **Ring not working**

Cause: The Token-Ring is beaconing. Recovery may take a few minutes. `Beaconing` displays in the Ring Status area.

Action: Wait for the message `010-202`, `010-203`, or `010-204` and follow the action for that message.

Example Data:

```
01/13/88  08:29:31 010-200 E  Ring not working
--------->  10005A001391 10005A00573E 00000000 0001

   Nearest Active        Beaconing    Beaconing    Error
   Upstream Neighbor     Adapter's    Adapter's    Count
   Adapter's Address     Address      Physical
                                      Address
```

A5ACG037

010-201 E Ring not working - this adapter beaconing

Cause: This adapter is beaconing. `Beaconing` displays in the Ring Status area for this message.

Action: Wait for the message `010-202`, `010-203`, or `010-204` and follow the action for that message.

Example Data:

```
01/14/88 07:45:32 010-201 E  Ring not working - this adapter beaconing
```

010-202 E Ring recovery failed

Cause: The adapter's auto recovery did not work. `Beaconing` displays in the Ring Status area.

Action: Manual recovery is required; see *Token-Ring Network Problem Determination Guide.*

Example Data:

```
01/15/88 10:33:32 010-202 E  Ring recovery failed
```

010-203 I Ring recovered

Cause: The Token-Ring recovered and is operating normally. `Normal` or `Soft Error` displays in the Ring Status area.

Action: This message is for information only.

Example Data:

```
01/13/88 08:29:56  010-203 I  Ring recovered
```

010-204 I Ring recovered - adapter removed

Cause: The adapter whose address is shown in the variable data is logically removed from the Token-Ring network. The Token-Ring network is operating normally without this adapter. If the reporting adapter and its nearest active upstream neighbor are both removed, two messages are given. `Normal` or `Soft Error` displays in the Ring Status area.

Action: See the *Token-Ring Network Problem Determination Guide* for additional information.

Example Data:

```
01/13/88  08:29:56 010-204 I  Ring recovered - adapter removed
--------->  10005A00573E
                 |
              Removed
              Adapter's
              Address
```

A5ACG038

010-215 E Ring adapter or lobe failed

Cause: The Token-Ring diagnostics determined that there was a wire fault. `Wire Fault` displays in the Ring Status area.

Action: See *Token-Ring Network Problem Determination Guide.*

Example Data:

```
02/05/88 05:59:35  010-215 E  Ring adapter or lobe failed
```

010-301 I Print file enabled

Cause: The **Print File** command is on, thus sending messages to a temporary file and to the log file.

Action: This message is for information only.

Example Data:

```
01/12/88 05:59:35  010-301 I  Print file enabled
```

010-302 I **Print file disabled and file queued**

Cause: The **Print File** command is off; the temporary file is queued to the printer and deleted.

Action: This message is for information only.

Example Data:

```
01/12/88 07:55:35  010-302 I  Print file disabled and file queued
```

010-303 E **Device ring queue full - data lost**

Cause: The Token-Ring VRM device driver has no free buffers available on the device ring queue. Frames are being received faster than the Token-Ring diagnostics can process them and data is being lost.

Action: When the rate of received frames decreases, normal message processing continues. If this message continues to display, use the **devices** command to increase the Token-Ring adapter **nobodr** parameter, number of buffers on device ring queue.

Example Data:

```
01/14/88 09:55:35  010-303 E  Device ring queue full - data lost
```

010-304 E **SLIH ring empty - data lost**

Cause: The SLIH ring queue used by the Token-Ring VRM device driver does not have an available buffer.

Action: You may want to run the **devices** command and try to increase the value of the Token-Ring adapter **norbosr** parameter, the total number of buffers on the SLIH ring queue. If the problem continues, try to balance the workload on the Token-Ring adapter or on the system.

Example Data:

```
01/14/88 09:55:35  010-304 E  SLIH ring empty - data lost
```

010-305 E Ring transmission error

Cause: A frame cannot be transmitted on the Token-Ring.

Action: Due to this error, it is possible that the Token-Ring diagnostics incorrectly reported the removal of a station from the Token-Ring.

Example Data:

```
01/14/88 09:55:35  010-305 E  Ring transmission error
```

010-306 E Error termination with error 010-*xxx*

Cause: The Token-Ring diagnostics ended due to error 010-*xxx*. This message is only written to the log file.

Action: See error 010-*xxx* in *IBM RT Messages Reference.*

Example Data:

```
01/12/88 07:01:56  010-306 E  Error termination with error 010-508
```

Ring Status Area

The Ring Status area displays the current status of the Token-Ring. The following messages display in the Ring Status area:

Normal The Token-Ring diagnostics is processing information, and the Token-Ring is operating normally.

Soft error The Token-Ring is experiencing intermittent failures.

Beaconing The Token-Ring diagnostics detects a hard error. This could be caused by a broken wire or a faulty adapter. The adapter address that is beaconing and the adapter address of the nearest active upstream neighbor display as variable data in the Data Messages Area.

Wire Fault There is a problem with the lobe between the attaching device and the access unit to which it is attached.

Token-Ring Diagnostics Status Area

The Token-Ring Diagnostics Status area displays temporary status information about the Token-Ring diagnostics. When the condition causing the temporary status changes, the message no longer displays. The following temporary messages display:

002 I Operation in progress - please wait

Figures

Glossary

access. To obtain data from or put data in storage.

access permission. A group of designations that determine who can access a particular AIX file and how the user may access the file.

account. The login directory and other information that give a user access to the system.

activity manager. A collection of system-supplied tasks allowing users to manage their activities. Provides the ability to list current activities (Activity List) and to begin, cancel, hide, and activate activities.

All Points Addressable (APA) display. A display that allows each pel to be individually addressed. An APA display allows for images to be displayed that are not made up of images predefined in character boxes. Contrast with *character display.*

allocate. To assign a resource, such as a disk file or a diskette file, to perform a specific task.

alphabetic. Pertaining to a set of letters a through z.

alphanumeric character. Consisting of letters, numbers and often other symbols, such as punctuation marks and mathematical symbols.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

American National Standards Institute. An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

application. A program or group of programs that apply to a particular business area, such as the Inventory Control or the Accounts Receivable application.

application program. A program used to perform an application or part of an application.

argument. Numbers, letters, or words that change the way a command works.

ASCII. See *American National Standard Code for Information Interchange.*

attribute. A characteristic. For example, the attribute for a displayed field could be blinking.

audit. To review and examine the activities of a data processing system mainly to test the adequacy and effectiveness of procedures for data privacy and data integrity.

audit bin. A file containing unprocessed audit records.

audit class. A list of events that define which actions taken on a system are recorded. They are defined by the system administrator in the user database.

audit event. An action (such as a command or access) taken on the system, which can be recorded by the system.

audit pipe. A chain of audit filter programs connected by pipes.

audit trail. For systems running in controlled access mode, a collection of events that could compromise system security recorded in the order in which they occurred.

auto carrier return. The system function that places carrier returns automatically within the text and on the display. This is accomplished by moving whole words that exceed the line end zone to the next line.

authentication. In computer security, a process used to verify the identity of the user, system, or protected resources.

backend. The program that sends output to a particular device. There are two types of backends: friendly and unfriendly.

background process. (1) A process that does not require operator intervention that can be run by the computer while the work station is used to do other work. (2) A mode of program execution in which the shell does not wait for program completion before prompting the user for another command.

backup copy. A copy, usually of a file or group of files, that is kept in case the original file or files are unintentionally changed or destroyed.

backup diskette. A diskette containing information copied from a fixed disk or from another diskette. It is used in case the original information becomes unusable.

bad block. A portion of a disk that can never be used reliably.

base address. The beginning address for resolving symbolic references to locations in storage.

base name. The last element to the right of a full path name. A filename specified without its parent directories.

batch printing. Queueing one or more documents to print as a separate job. The operator can type or revise additional documents at the same time. This is a background process.

batch processing. A processing method in which a program or programs process records with little or no operator action. This is a background process. Contrast with *interactive processing*.

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Involving a choice of two conditions, such as on-off or yes-no.

bit. Either of the binary digits 0 or 1 used in computers to store information. See also *byte*.

block. (1) A group of records that is recorded or processed as a unit. Same as *physical record*. (2) In data communications, a group of records that is recorded, processed, or sent as a unit. (3) A block is 512 bytes long. (4) A logical block is 2048 bytes long.

block file. A file listing the usage of blocks on a disk.

block special file. A special file that provides access to an input or output device is capable of supporting a file system. See also *character special file*.

bootstrap. A small program that loads larger programs during system initialization.

branch. In a computer program an instruction that selects one of two or more alternative sets of instructions. A conditional branch occurs only when a specified condition is met.

breakpoint. A place in a computer program, usually specified by an instruction, where execution may be interrupted by external intervention or by a monitor program.

buffer. (1) A temporary storage unit, especially one that accepts information at one rate and delivers it at another rate. (2) An area of storage, temporarily reserved for performing

input or output, into which data is read, or from which data is written.

burst pages. On continuous-form paper, pages of output that can be separated at the perforations.

byte. The amount of storage required to represent one character; a byte is 8 bits.

call. (1) To activate a program or procedure at its entry point. Compare with *load*.

callouts. An AIX kernel parameter establishing the maximum number of scheduled activities that can be pending simultaneously.

cancel. To end a task before it is completed.

carrier return. (1) In text data, the action causing line ending formatting to be performed at the current cursor location followed by a line advance of the cursor. Equivalent to the carriage return of a typewriter. (2) A keystroke generally indicating the end of a command line.

case sensitive. Able to distinguish between uppercase and lowercase letters.

character. A letter, digit, or other symbol.

character display. A display that uses a character generator to display predefined character boxes of images (characters) on the screen. This kind of display cannot address the screen any less than one character box at a time. Contrast with *All Points Addressable display*.

character key. A keyboard key that allows the user to enter the character shown on the key. Compare with *function keys*.

character position. On a display, each location that a character or symbol can occupy.

character set. A group of characters used for a specific reason; for example, the set of characters a printer can print or a keyboard can support.

character special file. A special file that provides access to an input or output device. The character interface is used for devices that do not use block I/O. See also *block special file*.

character string. A sequence of consecutive characters.

character variable. The name of a character data item whose value may be assigned or changed while the program is running.

child. (1) Pertaining to a secured resource, either a file or library, that uses the user list of a parent resource. A child resource can have only one parent resource. (2) In the AIX Operating System, child is a *process* spawned by a parent process that shares resources of parent process. Contrast with *parent*.

C language. A general-purpose programming language that is the primary language of the AIX Operating System.

class. Pertaining to the I/O characteristics of a device. AIX devices are classified as block or character.

client. A computer or process that accesses the data, services, or resources of another computer or process on the network.

close. (1) To end an activity and remove that window from the display.

code. (1) Instructions for the computer. (2) To write instructions for the computer; to *program*. (3) A representation of a condition, such as an error code.

code segment. See *segment*.

collating sequence. The sequence in which characters are ordered within the computer for sorting, combining, or comparing.

color display. A display device capable of displaying more than two colors and the shades produced via the two colors, as opposed to a monochrome display.

column. A vertical arrangement of text or numbers.

column headings. Text appearing near the top of columns of data for the purpose of identifying or titling.

command. A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command.

command interpreter. A program that sends instructions to the kernel; also called an interface.

command line. The area of the screen where commands are displayed as they are typed.

command line editing keys. Keys for editing the command line.

command programming language. Facility that allows programming by the combination of commands rather than by writing statements in a conventional programming language.

compile. (1) To translate a program written in a high-level programming language into a machine language program. (2) The computer actions required to transform a source file into an executable object file.

compress. (1) To move files and libraries together on disk to create one continuous area of unused space. (2) In data communications, to delete a series of duplicate characters in a character string.

computer virus. A program that can vandalize your files, although it usually does its defined task. It can spread itself to other files and directories on the system.

concatenate. (1) To link together. (2) To join two character strings.

concurrent group set. A list of the IDs of the various groups to which a user belongs.

condition. An expression in a program or procedure that can be evaluated to a value of either true or false when the program or procedure is running.

configuration. The group of machines, devices, and programs that make up a computer system. See also *system customization.*

configuration file. A file that specifies the characteristics of a system or subsystem, for example, the AIX queueing system.

consistent. Pertaining to a file system, without internal discrepancies.

console. (1) The main AIX display station. (2) A device name associated with the main AIX display station.

constant. A data item with a value that does not change. Contrast with *variable.*

context search. A search through a file whose target is a character string.

control block. A storage area used by a program to hold control information.

control commands. Commands that allow conditional or looping logic flow in shell procedures.

control program. Part of the AIX Operating System system that determines the order in which basic functions should be performed.

controlled cancel. The system action that ends the job step being run, and saves any new data already created. The job that is running can continue with the next job step.

copy. The action by which the user makes a whole or partial duplicate of already existing data.

crash. An unexpected interruption of computer service, usually due to a serious hardware or software malfunction.

current directory. The directory that is active, and can be displayed with the **pwd** command.

current line. The line on which the cursor is located.

current working directory. See *current directory*.

cursor. (1) A movable symbol (such as an underline) on a display, used to indicate to the operator where the next typed character will be placed or where the next action will be directed. (2) A marker that indicates the current data access location within a file.

cursor movement keys. The directional keys used to move the cursor.

customize. To describe (to the system) the devices, programs, users, and user defaults for a particular data processing system.

cylinder. All fixed disk or diskette tracks that can be read or written without moving the disk drive or diskette drive read/write mechanism.

daemon. See *daemon process*.

daemon process. A process begun by the root or the root shell that can be stopped only by the root. Daemon processes generally provide services that must be available at all times such as sending data to a printer.

database. A collection of information used for a specific purpose.

data block. See *block*.

data communications. The transmission of data between computers, or remote devices or both (usually over long distance).

data stream. All information (data and control information) transmitted over a data link.

dbm. Format for the Yellow Pages data base files.

debug. (1) To detect, locate, and correct mistakes in a program. (2) To find the cause of problems detected in software.

default. A value that is used when no alternative is specified by the operator.

default directory. The directory name supplied by the operating system if none is specified.

default drive. The drive name supplied by the operating system if none is specified.

default value. A value stored in the system that is used when no other value is specified.

delete. To remove. For example, to delete a file.

dependent work station. A work station having little or no stand-alone capability, that must be connected to a host or server in order to provide any meaningful capability to the user.

device. An electrical or electronic machine that is designed for a specific purpose and that attaches to your computer, for example, a printer, plotter, disk drive, and so forth.

device driver. A program that operates a specific device, such as a printer, disk drive, or display.

device name. A name reserved by the system that refers to a specific device.

diagnostic. Pertaining to the detection and isolation of an error.

diagnostic aid. A tool (procedure, program, reference manual) used to detect and isolate a device or program malfunction or error.

diagnostic routine. A computer program that recognizes, locates, and explains either a fault in equipment or a mistake in a computer program.

digit. Any of the numerals from 0 through 9.

directory. A type of file containing the names and controlling information for other files or other directories.

disable. To make nonfunctional.

discipline. Pertaining to the order in which requests are serviced, for example, first-come-first-served (fcfs) or shortest job next (sjn).

disk I/O. Fixed-disk input and output.

diskette. A thin, flexible magnetic plate that is permanently sealed in a protective cover. It can be used to store information copies from the disk or another diskette.

diskette drive. The mechanism used to read and write information on diskettes.

display device. An output unit that gives a visual representation of data.

display screen. The part of the display device that displays information visually.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator can see the information sent to or received from the computer.

dump. (1) To copy the contents of all or part of storage, usually to an output device. (2) Data that has been dumped.

dump diskette. A diskette that contains a dump or is prepared to receive a dump.

dump formatter. Program for analyzing a dump.

EBCDIC. See *extended binary-coded decimal interchange code.*

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC set.

edit. To modify the form or format of data.

edit buffer. A temporary storage area used by an editor.

editor. A program used to enter and modify programs, text, and other types of documents and data.

effective ID. The ID, either group or user, that is used to run a process. It can be set by a program to either the real or the saved ID.

emulation. Imitation; for example, when one computer imitates the characteristics of another computer.

enable. To make functional.

enter. To send information to the computer by pressing the **Enter** key.

entry. A single input operation on a work station.

environment. The settings for shell variables and paths associated with each process. These variables can be modified later by the user.

error-correct backspace. An editing key that performs editing based on a cursor position; the cursor is moved one position toward the beginning of the line, the character at the new cursor location is deleted, and all characters following the cursor are moved one position toward the beginning of the line (to fill the vacancy left by the deleted element).

escape character. A character that suppresses the special meaning of one or more characters that follow.

exit value. A numeric value that a command returns to indicate whether it completed successfully. Some commands return exit values that give other information, such as whether a file exists. Shell programs can test exit values to control branching and looping.

export. To make a resource, such as a file system, available to other machines on a network.

expression. A representation of a value. For example, variables and constants appearing alone or in combination with operators.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

feature. A programming or hardware option, usually available at an extra cost.

field. (1) An area in a record or panel used to contain a particular category of data. (2) The smallest component of a record that can be referred to by a name.

FIFO. See *first-in-first-out.*

file. A collection of related data that is stored and retrieved by an assigned name.

file name. The name used by a program to identify a file. See also *label.*

filename. In DOS, that portion of the file name that precedes the extension.

file specification (filespec). The name and location of a file. A file specification consists of a drive specifier, a path name, and a file name.

file system. The collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

filetab. An AIX kernel parameter establishing the maximum number of files that can be open simultaneously.

filter. A command that reads standard input data, modifies the data, and sends it to standard output.

filter programs. Programs designed to accept information from input, process the data, and write the results to standard output.

first-in-first-out (FIFO). A named permanent pipe. A FIFO allows two unrelated processes to exchange information using a pipe connection.

first level interrupt handler (FLIH). A routine that receives control of the system as a result of a hardware interrupt. One FLIH is assigned to each of the six interrupt levels.

fixed disk. A flat, circular, nonremoveable plate with a magnetizable surface layer on which data can be stored by magnetic recording.

fixed-disk drive. The mechanism used to read and write information on fixed disk.

flag. A modifier that appears on a command line with the command name that defines the action of the command. Flags in the AIX Operating System almost always are preceded by a dash.

font. A family or assortment of characters of a given size and style.

foreground. A mode of program execution in which the shell waits for the program specified on the command line to complete before returning your prompt.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) The pattern which determines how data is recorded.

formatted diskette. A diskette on which control information for a particular computer system has been written but which may or may not contain any data.

free list. A list of available space on each file system. This is sometimes called the free-block list.

free-block list. See *free list.*

full path name. The name of any directory or file expressed as a string of directories and files beginning with the root directory.

function. A synonym for procedure. The C language treats a function as a data type that

contains executable code and returns a single value to the calling function.

function keys. Keys that request actions but do not display or print characters. Included are the keys that normally produce a printed character, but when used with the code key produce a function instead. Compare with *character key*.

generation. For some remote systems, the translation of configuration information into machine language.

Gid. See *group number*.

global. Pertains to information available to more than one program or subroutine.

global action. An action having general applicability, independent of the context established by any task.

global character. The special characters * and ? that can be used in a file specification to match one or more characters. For example, placing a ? in a file specification means any character can be in that position.

global search. The process of having the system look through a document for specific characters, words, or groups of characters.

global variable. A symbol defined in one program module, but used in other independently assembled program modules.

graphic character. A character that can be displayed or printed.

group name. A name that uniquely identifies a group of users to the system.

group number (Gid). A unique number assigned to a group of related users. The group number can often be substituted in commands that take a group name as an argument.

hardware. The equipment, as opposed to the programming, of a computer system.

header. Constant text that is formatted to be in the top margin of one or more pages.

header label. A special set of records on a diskette describing the contents of the diskette.

here document. Data contained within a shell program or procedure (also called *inline input*).

highlight. To emphasize an area on the display by any of several methods, such as brightening the area or reversing the color of characters within the area.

history file. A file containing a log of system actions and operator responses.

hog factor. In system accounting, an analysis of how many times each command was run, how much processor time and memory it used, and how intensive that use was.

home directory. (1) A directory associated with an individual user. (2) The user's current directory on login or after issuing the **cd** command with no argument.

I/O. See *input/output*.

ID. Identification.

IF expressions. Expressions within a procedure, used to test for a condition.

indirect block. A block containing pointers to other blocks. Indirect blocks can be single-indirect, double-indirect, or triple-indirect.

informational message. A message providing information to the operator, that does not require a response.

initial program load (IPL). The process of loading the system programs and preparing the system to run jobs. See *initialize*.

initialize. To set counters, switches, addresses, or contents of storage to zero or other starting

values at the beginning of, or at prescribed points in, the operation of a computer routine.

inline input. See *here document.*

i-node. The internal structure for managing files in the system. I-nodes contain all of the information pertaining to the node, type, owner, and location of a file. A table of i-nodes is stored near the beginning of a file system.

i-number. A number specifying a particular i-node on a file system.

inodetab. A kernel parameter that establishes the size of a table in memory for storing copies of i-nodes for all active files.

input. Data to be processed.

input device. Physical devices used to provide data to a computer.

input file. A file opened by a program so that the program can read from that file.

input list. A list of variables to which values are assigned from input data.

input redirection. The specification of an input source other than the standard one.

input-output device number (IODN). A value assigned to a device by the virtual machine or to a virtual device by the Virtual Resource Manager. This number uniquely identifies the device regardless of whether it is real or virtual.

input-output file. A file opened for input and output use.

input/output (I/O). Pertaining to either input, output, or both between a computer and a device.

input/output subsystem. The part of the VRM comprised of processes and device managers that provides the mechanisms for data transfer and I/O device management and control.

interactive processing. A processing method in which each system user action causes response from the program or the system. Contrast with *batch processing.*

interface. A shared boundary between two or more entities. An interface might be a hardware component to link two devices together or it might be a portion of storage or registers accessed by two or more computer programs.

interleave factor. Specification of the ratio between contiguous physical blocks (on a fixed-disk) and logically contiguous blocks (as in a file).

interrupt. (1) To temporarily stop a process. (2) In data communications, to take an action at a receiving station that causes the sending station to end a transmission. (3) A signal sent by an I/O device to the processor when an error has occurred or when assistance is needed to complete I/O. An interrupt usually suspends execution of the currently executing program.

IPL. See *initial program load.*

job. (1) A unit of work to be done by a system. (2) One or more related procedures or programs grouped into a procedure.

job queue. A list, on disk, of jobs waiting to be processed by the system.

justify. To print a document with even right and left margins.

kbuffers. An AIX kernel parameter establishing the number of buffers that can be used by the kernel.

K-byte. See *kilobyte.*

kernel. A part of the AIX Operating System which participates in the control of computer functions such as input/output, management and control of hardware and software, and scheduling of user processes.

kernel parameters. Variables that specify how the kernel allocates certain system resources.

key pad. A physical grouping of keys on a keyboard (for example, numeric key pad, and cursor key pad).

keyboard. An input device consisting of various keys allowing the user to input data, control cursor and pointer locations, and to control the dialog between the user and the display station

keylock feature. A security feature in which a lock and key can be used to restrict the use of the display station.

keyword. One of the predefined words of a programming language; a reserved word.

keyword argument. One type of variable assignment that can be made on the command line.

kill. An AIX Operating System command that stops a process.

kill character. The character that is used to delete a line of characters entered after the user's prompt.

kilobyte. 1024 bytes.

kprocs. An AIX kernel parameter establishing the maximum number of processes that the kernel can run simultaneously.

label. (1) The name in the disk or diskette volume table of contents that identifies a file. See also *file name.* (2) The field of an instruction that assigns a symbolic name to the location at which the instruction begins, or such a symbolic name.

left margin. The area on a page between the left paper edge and the leftmost character position on the page.

left-adjust. The process of aligning lines of text at the left margin or at a tab setting such that the leftmost character in the line or filed is in the leftmost position. Contrast with *right-adjust.*

library. A collection of functions, calls, subroutines, or other data.

licensed program product (LPP). Software programs that remain the property of the manufacturer, for which customers pay a license fee.

line editor. An editor that modifies the contents of a file one line at a time.

linefeed. An ASCII character that causes an output device to move forward one line.

link. A connection between an i-node and one or more file names associated with it.

literal. A symbol or a quantity in a source program that is itself data, rather than a reference to data.

load. (1) To move data or programs into storage. (2) To place a diskette into a diskette drive, or a magazine into a diskette magazine drive. (3) To insert paper into a printer.

loader. A program that reads run files into main storage, thus preparing them for execution.

local. Pertaining to a device directly connected to your system without the use of a communications line. Contrast with *remote.*

log. To record; for example, to log all messages on the system printer. A list of this type is called a log, such as an error log.

log in. To begin a session at a display station.

log in shell. The program, or command interpreter, started for a user at log in.

log off. To end a session at a display station.

log out. To end a session at a display station.

logical device. A file for conducting input or output with a physical device.

login User ID. The ID set by the system for a user at login.

loop. A sequence of instructions performed repeatedly until an ending condition is reached.

main storage. The part of the processing unit where programs are run.

maintenance system. A special version of the AIX Operating System which is loaded from diskette and used to perform system management tasks.

major device number. A system identification number for each device or type of device.

mapped files. Files on the fixed-disk that are accessed as if they are in memory.

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters.

matrix. An array arranged in rows and columns.

maxprocs. A kernel parameter establishing the maximum number of processes that can be run simultaneously by a user.

memory. Storage on electronic chips. Examples of memory are random access memory, read only memory, or registers. See *storage.*

menu. A displayed list of items from which an operator can make a selection.

message. (1) A response from the system to inform the operator of a condition which may affect further processing of a current program. (2) Information sent from one user in a multiuser operating system to another.

minidisk. A logical division of a fixed disk.

minor device number. A number used to specify various types of information about a particular device, for example, to distinguish among several printers of the same type.

mode word. An i-node field that describes the type and state of the i-node.

modem. See *modulator-demodulator.*

modulation. Changing the frequency or size of one signal by using the frequency or size of another signal.

modulator-demodulator (modem). A device that converts data from the computer to a signal that can be transmitted on a communications line, and converts the signal received to data for the computer.

module. (1) A discrete programming unit that usually performs a specific task or set of tasks. Modules are subroutines and calling programs that are assembled separately, then linked to make a complete program. (2) See *load module.*

mount. To make a file system accessible.

mountab. A kernel parameter establishing the maximum number of file systems that can be mounted simultaneously.

multiprogramming. The processing of two or more programs at the same time.

multivolume file. A diskette file occupying more than one diskette.

nest. To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one loop (the nested loop) within another loop (the nesting loop); to nest one subroutine (the nested subroutine) within another subroutine (the nesting subroutine).

network. A collection of products connected by communication lines for information exchange between locations.

Network File System (NFS). A licensed program that allows users to share files

between machines in a network environment by mounting the file systems located on remote machines.

new-line character. A control character that causes the print or display position to move to the first position on the next line.

null. Having no value, containing nothing.

null character (NUL). The character hex 00, used to represent the absence of a printed or displayed character.

numeric. Pertaining to any of the digits 0 through 9.

object code. Machine-executable instruction, usually generated by a compiler from source code written in a higher level language. consists of directly executable machine code. For programs that must be linked, object code consists of relocatable machine code.

octal. A base eight numbering system.

open. (1) To make a file available to a program for processing.

operating system. Software that directs and controls the hardware and software in the computer system in which the operating system resides, by providing services such as resource allocation, scheduling, input/output control, and data management.

operation. A specific action (such as move, add, multiply, load) that the computer performs when requested.

operator. A symbol representing an operation to be done.

output. The result of processing data.

output devices. Physical devices used by a computer to present data to a user.

output file. A file that is opened by a program so that the program can write to that file.

output redirection. The specification of an output destination other than the standard one.

override. (1) A parameter or value that replaces a previous parameter or value. (2) To replace a parameter or value.

overwrite. To write output into a storage or file space that is already occupied by data.

owner. The user who has the highest level of access authority to a data object or action, as defined by the object or action.

pad. To fill unused positions in a field with dummy data, usually zeros or blanks.

page. A block of instructions, data, or both.

page space minidisk. The area on a fixed disk that temporarily stores instructions or data currently being run. See also *minidisk.*

pagination. The process of adjusting text to fit within margins and/or page boundaries.

paging. The action of transferring instructions, data, or both between real storage and external page storage.

parallel processing. The condition in which multiple tasks are being performed simultaneously within the same activity.

parameter. Information that the user supplies to a panel, command, or function.

parent. Pertaining to a secured resource, either a file or library, whose user list is shared with one or more other files or libraries. Contrast with *child.*

parent directory. The directory one level above the current directory.

partition. See *minidisk.*

password. A string of characters that, when entered along with a user identification, allows an operator to sign on to the system.

password security. A program product option that helps prevent the unauthorized use of a display station, by checking the password entered by each operator at log in.

path name. See *full path name* and *relative path name.*

pattern-matching character. Special characters such as * or ? that can be used in search patterns. Some used in a file specification to match one or more characters. For example, placing a ? in a file specification means any character can be in that position. Pattern-matching characters are also called wildcards.

permission code. A three-digit octal code, or a nine-letter alphabetic code, indicating the access permissions. The access permissions are read, write, and execute.

permission field. One of the three-character fields within the permissions column of a directory listing indicating the read, write, and run permissions for the file or directory owner, group, and all others.

phase. One of several stages file system checking and repair performed by the **fsck** command.

physical device. See *device.*

physical file. An indexed file containing data for which one or more alternative indexes have been created.

physical record. (1) A group of records recorded or processed as a unit. Same as *block.* (2) A unit of data moved into or out of the computer.

PID. See *process ID.*

pipe. To direct the data so that the output from one process becomes the input to another process.

pipeline. A direct, one-way connection between two or more processes.

pitch. A unit of width of typewriter type, based on the number of times a letter can be set in a linear inch. For example, 10-pitch type has 10 characters per inch.

platen. The support mechanism for paper on a printer, commonly cylindrical, against which printing mechanisms strike to produce an impression.

pointer. A logical connection between physical blocks.

port. (1) To make the programming changes necessary to allow a program that runs on one type of computer to run on another type of computer. (2) An access point for data input to or data output from a computer system. See *connector*

portmap (portmapper). A daemon process that matches RPC port numbers to RPC services provided by NFS servers to conduct remote services in the NFS.

position. The location of a character in a series, as in a record, a displayed message, or a computer printout.

positional parameter. A shell facility for assigning values from the command line to variables in a program.

print queue. A file containing a list of the names of files waiting to be printed.

printout. Information from the computer produced by a printer.

priority. The relative ranking of items. For example, a job with high priority in the job queue will be run before one with medium or low priority.

priority number. A number that establishes the relative priority of printer requests.

privileged instructions. System control instructions that can only run in the processor's privileged state (VRM mode).

Privileged instructions generally manipulate virtual machines or the memory manager; they typically are not used by application programmers. See *privileged state.*

privileged state. A hardware protection state in which the processor can run privileged instructions.

privileged user. The account with superuser authority.

problem determination. The process of identifying why the system is not working. Often this process identifies programs, equipment, data communications facilities, or user errors as the source of the problem.

problem determination procedure. A prescribed sequence of steps aimed at recovery from, or circumvention of, problem conditions.

procedure. See *shell procedure.*

process. (1) A sequence of actions required to produce a desired result. (2) An entity receiving a portion of the processor's time for executing a program. (3) An activity within the system begun by entering a command, running a shell program, or being started by another process.

process accounting. An analysis of the use each process makes of the processing unit, memory, and I/O resources.

process ID (PID). A unique number assigned to a process that is running.

profile. (1) A file containing customized settings for a system or user (2) Data describing the significant features of a user, program, or device.

program. A file containing a set of instructions conforming to a particular programming language syntax.

prompt. A displayed request for information or operator action.

propagation time. The time necessary for a signal to travel from one point on a communications line to another.

qdaemon. The daemon process that maintains a list of outstanding jobs and sends them to the specified device at the appropriate time.

queue. A line or list formed by items waiting to be processed.

queued message. A message from the system that is added to a list of messages stored in a file for viewing by the user at a later time. This is in contrast to a message that is sent directly to the screen for the user to see immediately.

quit. A key, command, or action that tells the system to return to a previous state or stop a process.

quote. To mask the special meaning of certain characters; to cause them to be taken literally.

random access. An access mode in which records can be read from, written to, or removed from a file in any order.

read only. Pertaining to file system mounting, a condition that allows data to be read, but not modified.

real ID. The ID (user or group) that is set at system check time.

recovery procedure. (1) An action performed by the operator when an error message appears on the display screen. Usually, this action permits the program to continue or permits the operator to run the next job. (2) The method of returning the system to the point where a major system error occurred and running the recent critical jobs again.

redirect. To divert data from a process to a file or device to which it would not normally go.

reference count. In an i-node, a record of the total number of directory entries that refer to the i-node.

relational expression. A logical statement describing the relationship (such as greater than or equal) of two arithmetic expressions or data items.

relational operator. The reserved words or symbols used to express a relational condition or a relational expression.

relative address. An address specified relative to the address of a symbol. When a program is relocated, the addresses themselves will change, but the specification of relative addresses remains the same.

relative addressing. A means of addressing instructions and data areas by designating their locations relative to some symbol.

relative path name. The name of a directory or file expressed as a sequence of directories followed by a file name, beginning from the current directory.

remote. Pertaining to a system or device that is connected to your system through a communications line. Contrast with *local.*

remote procedure call. A request for a service located on another computer in the network.

Remote Procedure Call. The interface NFS uses for remote procedure calls.

reserved character. A character or symbol that has a special (nonliteral) meaning unless quoted.

reserved word. A word that is defined in a programming language for a special purpose, and that must not appear as a user-declared identifier.

reset. To return a device or circuit to a clear state.

restore. To return to an original value or image. For example, to restore a library from diskette.

right adjust. The process of aligning lines of text at the right margin or tab setting such that the rightmost character in the line or file is in the rightmost position.

right justify. See right align.

right margin. The area on a page between the last text character and the right upper edge.

right-adjust. To place or move an entry in a field so that the rightmost character of the field is in the rightmost position. Contrast with *left-adjust.*

root. Another name sometimes used for superuser.

root directory. The top level of a tree-structured directory system.

root file system. The basic AIX Operating System file system, which contains operating system files and onto which other file systems can be mounted. The root file system is the file system that contains the files that are run to start the system running.

routine. A set of statements in a program causing the system to perform an operation or a series of related operations.

run. To cause a program, utility, or other machine function to be performed.

run-time environment. A collection of subroutines and shell variables that provide commonly used functions and information for system components.

scratch file. A file, usually used as a work file, that exists until the program that uses it ends.

screen. See *display screen.*

scroll. To move information vertically or horizontally to bring into view information that is outside the display screen boundaries.

second level interrupt handler (SLIH). A routine that handles the processing of an

interrupt from a specific adapter. An SLIH is called by the first level interrupt handler associated with that interrupt level.

sector. (1) An area on a disk track or a diskette track reserved to record information. (2) The smallest amount of information that can be written to or read from a disk or diskette during a single read or write operation.

secure attention key (SAK). A key sequence which, when pressed, provides exclusive, trusted access to the terminal. All processes associated with the terminal are ended, and all accesses to the terminal are revoked.

security. The protection of data, system operations, and devices from accidental or intentional ruin, damage, or exposure.

segment. A contiguous area of virtual storage allocated to a job or system task. A program segment can be run by itself, even if the whole program is not in main storage.

separator. A character used to separate parts of a command or file.

server. A computer or process that provides data, services, or resources that can be accessed by other computers or processes in the network.

sequential access. An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file.

session records. In the accounting system, a record of time connected and line usage for connected display stations, produced from log in and log out records.

set flags. Flags that can be put into effect with the shell set command.

shared printer. A printer that is used by more than one work station.

shell. See *shell program.*

shell procedure. A series of commands combined in a file that carry out a particular function when the file is run or when the file is specified as an argument to the sh command. Shell procedures are frequently called shell scripts.

shell program. A program that accepts and interprets commands for the operating system (there is an AIX shell program and a DOS shell program).

shell prompt. The character string on the command line indicating that the system can accept a command (typically the $ character).

shell script. See *shell* procedure.

shell variables. Facilities of the shell program for assigning variable values to constant names.

size field. In an i-node, a field that indicates the size, in bytes, of the file associated with the i-node.

software. Programs.

sort. To rearrange some or all of a group of items based upon the contents or characteristics of those items.

source diskette. The diskette containing data to be copied, compared, restored, or backed up.

source program. A set of instructions written in a programming language, that must be translated to machine language compiled before the program can be run.

special character. A character other than an alphabetic or numeric character. For example; *, +, and % are special characters.

special file. Special files are used in the AIX system to provide an interface to input/output devices. There is at least one special file for each device connected to the computer. Contrast with *directory* and *file.* See also *block special file* and *character special file.*

spool files. Files used in the transmission of data among devices.

standalone shell. A limited version of the shell program used for system maintenance.

standalone work station. A work station that can be used to preform tasks independent of (without being connected to) other resources such as servers or host systems.

standard error. The place where many programs place error messages.

standard input. The primary source of data going into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

standard output. The primary destination of data coming from a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can be to a file or another command.

stanza. A group of lines in a file that together have a common function. Stanzas are usually separated by blank lines, and each stanza has a name.

statement. An instruction in a program or procedure.

status. (1) The current condition or state of a program or device. For example, the status of a printer. (2) The condition of the hardware or software, usually represented in a status code.

storage. (1) The location of saved information. (2) In contrast to memory, the saving of information on physical devices such as disk or tape. See *memory*.

storage device. A device for storing and/or retrieving data.

string. A linear sequence of entities such as characters or physical elements. Examples of strings are alphabetic string, binary element string, bit string, character string, search string, and symbol string.

su. See *superuser*.

subdirectory. A directory contained within another directory in the file system hierarchy.

subprogram. A program invoked by another program, such as a subshell.

subroutine. (1) A sequenced set of statements that may be used in one or more computer programs and at one or more points in a computer program. (2) A routine that can be part of another routine.

subscript. An integer or variable whose value refers to a particular element in a table or an array.

subshell. An instance of the shell program started from an existing shell program.

substring. A part of a character string.

subsystem. A secondary or subordinate system, usually capable of operating independently of, or synchronously with, a controlling system.

superblock. The most critical part of the file system containing information about every allocation or deallocation of a block in the file system.

superuser (su). The user who can operate without the restrictions designed to prevent data loss or damage to the system (User ID 0).

superuser authority. The unrestricted ability to access and modify any part of the operating system associated with the user who manages the system. The authority obtained when one logs in as **root**.

symbolic link. A type of file that contains the path name to another file or directory; it functions as a pointer to the other file or directory.

system. The computer and its associated devices and programs.

system call. A request by an active process for a service by the system kernel.

system customization. A process of specifying the devices, programs, and users for a particular data processing system.

system date. The date assigned by the system user during setup and maintained by the system.

system dump. A copy of memory from all active programs (and their associated data) whenever an error stops the system. Contrast with *task dump.*

system management. The tasks involved in maintaining the system in good working order and modifying the system to meet changing requirements.

system parameters. See *kernel parameters.*

system profile. A file containing the default values used in system operations.

system unit. The part of the system that contains the processing unit, the disk drives, and the diskette drives.

system user. A person who uses a computer system.

target diskette. The diskette to be used to receive data from a source diskette.

task. A basic unit of work to be performed. Examples are a user task, a server task, and a processor task.

task dump. A copy of memory from a program that failed (and its associated data). Contrast with *system dump.*

terminal. An input/output device containing a keyboard and either a display device or a printer. Terminals usually are connected to a computer and allow a person to interact with the computer.

text. A type of data consisting of a set of linguistic characters (for example, alphabet, numbers, and symbols) and formatting controls.

text application. A program defined for the purpose of processing text data (for example, memos, reports, and letters).

text editing program. See *editor* and *text application.*

texttab. A kernel parameter establishing the size of the text table, in memory, that contains one entry for each active shared program text segment.

trace. To record data that provides a history of events occurring in the system.

trace table. A storage area into which a record of the performance of computer program instructions is stored.

track. A circular path on the surface of a fixed disk or diskette on which information is magnetically recorded and from which recorded information is read.

trap. An unprogrammed, hardware-initiated jump to a specific address. Occurs as a result of an error or certain other conditions.

tree-structured directories. A method for connecting directories such that each directory is listed in another directory except for the root directory, which is at the top of the tree.

trojan horse. A program that can vandalize your files, although it does its defined task.

truncate. To shorten a field or statement to a specified length.

trusted communications path. A secure path to the system, invoked with a key sequence and used when entering or changing security-relevant information in the system. For example, when changing passwords or logging in to the system.

trusted computing base. The total of all system components, both hardware and software, that protect data in the system.

trusted program. A program known to be free of trojan horses and computer viruses and which assures proper function.

trusted shell. A modified command interpreter that provides a restricted environment to perform administrative tasks in a secure manner.

typematic key. A key that repeats its function multiple times when held down.

typestyle. Characters of a given size, style and design.

Uid. See *user number.*

update. An improvement for some part of the system.

user. The name associated with an account.

user account. See *account.*

user ID. See *user number.*

user name. A name that uniquely identifies a user to the system.

user number (Uid). (1) A unique number identifying an operator to the system. This string of characters limits the functions and information the operator is allowed to use. The Uid can often be substituted in commands that take a user's name as an argument.

user profile. A file containing a description of user characteristics and defaults (for example, printer assignment, formats, group ID) to be conveyed to the system while the user is signed on.

utility. A service; in programming, a program that performs a common service function.

valid. (1) Allowed. (2) True, in conforming to an appropriate standard or authority.

value. (1) In Usability Services, information selected or typed into a pop-up. (2) A set of characters or a quantity associated with a parameter or name. (3) In programming, the contents of a storage location.

variable. A name used to represent a data item whose value can change while the program is running. Contrast with *constant.*

verify. To confirm the correctness of something.

version. Information in addition to an object's name that identifies different modification levels of the same logical object.

virtual device. A device that appears to the user as a separate entity but is actually a shared portion of a real device. For example, several virtual terminals may exist simultaneously, but only one is active at any given time.

virtual machine. The hardware-independent portion (kernel, shells, libraries, and other subsystems) of the AIX Operating System and user applications.

virtual machine interface (VMI). A standard software interface between the kernel and the VRM.

Virtual Resource Manager (VRM). A portion of the ¯ that provides various services, interfaces, and runtime routines, through which AIX controls the IBM RT hardware and peripherals.

virtual resources. See *virtual resource manager.*

virtual storage. Addressable space that appears to be real storage. From virtual storage, instructions and data are mapped into real storage locations.

virtual terminal. Any of several logical equivalents of a display station available at a single physical display station.

Volume ID (Vol ID). A series of characters recorded on the diskette used to identify the diskette to the user and to the system.

VRM. See *virtual resource manager.*

wildcard. See *pattern-matching characters.*

word. A contiguous series of 32 bits (4 bytes) in storage, addressable as a unit. The address of the first byte of a word is evenly divisible by four.

work file. A file used for temporary storage of data being processed.

work station. A device at which an individual may transmit information to, or receive information from, a computer for the purpose of performing a task, for example, a display station or printer. See *programmable work station* and *dependent work station.*

working directory. See *current directory.*

wrap around. Movement of the point of reference in a file from the end of one line to the beginning of the next, or from one end of a file to the other.

XDR (eXternal Data Representation). A protocol used to represent the objects in machine data externally. The internal data representations of various machine types are represented in a uniform format so that networked machines can communicate regardless of their manufacturer or structure algoritm.

Yellow Pages (YP). A network service that utilizes a centralized data base system to administer system information, such as passwords and machine names. The YP is installed with the NFS licensed program.

Index

Special Characters

A

B

D

E

F

K

L

M

N

O

P

Q

R

S

U

Numerics

Managing the AIX Operating System — Book Title

SC23-2008-1 — Order No.

Book Evaluation Form

Your comments can help us produce better books. You may use this form to communicate your comments about this book, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Please take a few minutes to evaluate this book as soon as you become familiar with it. Circle Y (Yes) or N (No) for each question that applies and give us any information that may improve this book.

Y N Is the purpose of this book clear?

Y N Is the table of contents helpful?

Y N Is the index complete?

Y N Are the chapter titles and other headings meaningful?

Y N Is the information organized appropriately?

Y N Is the information accurate?

Y N Is the information complete?

Y N Is only necessary information included?

Y N Does the book refer you to the appropriate places for more information?

Y N Are terms defined clearly?

Y N Are terms used consistently?

Y N Are the abbreviations and acronyms understandable?

Y N Are the examples clear?

Y N Are examples provided where they are needed?

Y N Are the illustrations clear?

Y N Is the format of the book (shape, size, color) effective?

Other Comments

What could we do to make this book or the entire set of books for this system easier to use?

Optional Information

Your name ______

Company name ______

Street address ______

City, State, ZIP ______

No postage necessary if mailed in the U.S.A.

Tape

Please Do Not Staple

Tape

Cut or Fold Along Line

Fold and Tape

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758-3493

The IBM AIX/RT Family

Reader's Comment Form

Managing the AIX Operating System

SC23-2008-1

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact your IBM representative, your IBM authorized dealer, or your IBM authorized remarketer.

Comments:

Tape

Please Do Not Staple

Tape

Cut or Fold Along Line

Fold and Tape

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758-3493

International Business
Machines Corporation
11400 Burnet Road
Austin, Texas 78758

Printed in the
United States of America

SC23-2008-1

SC23-2008-1

27F4353